Diogo Phelipe Busanello da Silva

UMA ABORDAGEM PARA COMPOSIÇÃO SEMIAUTOMÁTICA DE SERVIÇOS BASEADA EM SUGESTÕES SEMÂNTICAS

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina para a obtenção do Grau de mestre em Ciência da Computação
Orientador: Prof. Dr. Frank A. Siqueira

Florianópolis
2015

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Silva, Diogo Phelipe Busanello da
Uma Abordagem para Composição Semiautomáitca de Serviços Baseada em Sugestões Semânticas / Diogo Phelipe Busanello da Silva ; orientador, Frank Augusto Siqueira - Florianópolis, SC, 2015.
94 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro Tecnológico. Programa de Pós-Graduação em Ciência da Computação.

Inclui referências

1. Ciência da Computação. 2. Web Services Semânticos. 3. Composição de Serviço. 4. SAWSDL. I. Siqueira, Frank Augusto. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Ciência da Computação. III. Título.

Diogo Phelipe Busanello da Silva

UMA ABORDAGEM PARA COMPOSIÇÃO SEMIAUTOMÁTICA DE SERVIÇOS BASEADA EM SUGESTÕES SEMÂNTICAS

Esta Dissertação foi julgada adequada para obtenção do Título de "Mestre em Ciência da Computação", e aprovada em sua forma final pelo Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Santa Catarina.

Florianópolis, 11 de junho de 2015.

_____
Prof. Ronaldo dos Santos Mello, Dr.
Coordenador do Programa

Banca Examinadora:

_____
Prof. Frank Augusto Siqueira, Dr.
Orientador
Universidade Federal de Santa Catarina

_____
Prof. Eros Comunello, Dr.
Universidade do Vale do Itajaí

_____
Prof. Mario Antonio Ribeiro Dantas, Dr.
Universidade Federal de Santa Catarina

_____
Prof. Renato Fileto, Dr.
Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus pais e amigos, que de alguma forma me ajudaram a chegar ao fim desta jornada.

AGRADECIMENTOS

Este é o fim da jornada que teve início em 2012, devo a conclusão dessa etapa da minha vida às diversas pessoas que fizeram parte da minha vida nesses últimos três anos.

Primeiramente, gostaria de agradecer meu orientador, Frank Siqueira, que acreditou em meu potencial ao me conceder a oportunidade de cursar este programa de pós-graduação. Também por sua paciência e serenidade ao lidar com meus atrasos e compreensão com meus "bloqueios" criativos ao escrever esta dissertação. A conclusão deste trabalho não seria possível sem isso.

Aos senhores José Itamar da Silva e Nadir Busanello, também conhecidos nas horas vagas por pai e mãe, por serem meus grandes exemplos de profissionalismo e meus modelos de como ser ou não ser (eis a questão) como pessoa. Sem o apoio incondicional de meus pais eu não teria chegado até aqui. Vocês sempre serão parte importante de minhas conquistas.

Aos meus amigos que sempre estiveram ao meu lado nos momentos mais difíceis desta jornada. Um agradecimento especial à Raquel Sitjar por seus conselhos, bons e ruins. Ao Lucas Alencar por partilhar das mesmas experiências desde a graduação e melhor entender o que estava passando. À Poliana Corrêa pelas horas de conversas sobre os assuntos mais diversos, que me ajudava a distrair dos outros problemas. Ao Cadi Calegari pelo período que dividimos o apartamento em Florianópolis e por não desistir de tentar me levar para a praia nos finais de semana, conseguindo a marca impressionante de conseguir me arrastar três vezes durante um ano.

Gostaria de agradecer ao pessoal do LaPeSD, pela recepção e amizade durante o período alocado em Florianópolis. E também o pessoal da Nokia, em especial o grupo do "Haiti" pelas conversas divertidas e algumas vezes bizarras durante o horário (prolongado) de almoço e durante os cafezinhos na copa.

Por último, gostaria de agradecer ao Chucky por suas incontáveis tentativas de contribuir com minha dissertação gerando os textos mais incompreensíveis possíveis em "gatês" toda vez que eu me ausentava do computador, e me ensinando a importância de salvar constantemente os documentos.

"*I may not have gone where I intended to go, but I think I have ended up where I needed to be.*"
Douglas Adams

RESUMO

O uso de semântica na Web tem o intuito de permitir que as informações possam sem processadas por máquinas, geralmente através de significados definidos por ontologias, que são uma especificação formal e explícita de um conceito compartilhado. Com Web Services é possível a invocação de funcionalidade de softwares através de interfaces bem definidas. A utilização de semântica para descrição de Web Services possibilita ainda a automação, total ou parcial, do processo de criação de composições de serviços, que mostra-se necessária quando um único serviço não consegue prover toda a funcionalidade desejada pelo cliente. Na literatura existem vários esforços para buscar automação no processo de composição de serviços com base na utilização de descrição semântica. A completa automação do processo de composição de serviço vem associada ao alto custo computacional e à exclusão do usuário do processo. Esse trabalho apresenta uma abordagem hibrida, semiautomática, chamada de S^3M, na qual há participação do usuário no processo de composição de serviços, somada a sugestões de operações que são compatíveis com a composição sendo criado. Isso é feito através do *match* semântico entre as entradas das operações candidatas junto às saídas das operações contidas na composição sendo criada, somada a outras métricas. Como a qualidade da sugestão depende diretamente das ontologias associadas ao serviços, a avaliação da proposta ficou limitada a testes de desempenho e simulação do comportamento do algoritmo em determinados cenários.
Palavras-chave: Web Services Semânticos. Composição de Serviços. SAWSDL.

ABSTRACT

The use of semantic in the Web aims to make possible the processing of information by machines, usually through a well defined meaning in ontologies, i.e. a formal explicit specification of a shared conceptualization. With Web Service it is possible the invocacional of software functionality through interfaces. Semantic is used to add meaning to Web Services operations, inputs and outputs. The use of semantic description in Web Services also allows either full or partial automation of the service composition creation process, which is necessary when a single service cannot provide all features required by the client. There are several efforts in literature seeking automation of the service composition process based on the use of semantic descriptions. The complete automation on the service composition process results in a high computational cost and in excluding the user from the process. This work presents a hybrid (i.e. semi-automatic) approach, called S^3M. In which the user takes part in the service composition process, receiving suggestions of services that are compatible with the composition during its creation. This is made through the semantic match between the inputs of the candidates operations and the outputs of the operations inside the composition been designed, with another metrics. The quality of the suggestion are directly dependent with the ontologies associated with the services, making the evaluation of the presented work limited to performance tests and simulation, where the behavior of de suggestion mechanism could be analyzed.

Keywords: Semantic Web Services. Web Service Composition. SAWSDL.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

HTML: HyperText Markup Language
OWL: Web Ontology Language
OWL-S: Web Ontology Language for Services
QoS: Quality of Service
RDF: Resource Description Framework
RDFS: Resource Description Framework Schema
SAWDL: Samantics Annotations for WSDL
SOA: Service Oriented Architecture
SOAP: Simple Object Access Protocol
SWS: Semantic Web Service
UDDI: Universal Description, Discovery, and Integration
URI : Uniform Resource Identifier
W3C: World Wide Web Consortium
WSDL: Web Services Description Language
WSML: Web Service Modeling Language
WSMO: Web Service Modeling Ontology
XML: Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Em 2001, Tim Berners-Lee propôs o que seria a evolução da Web da época, que consistia basicamente de informação com significado apenas para os humanos. Essa nova Web seria uma extensão da anterior, onde as informações contidas deixariam de ter significado apenas para os humanos e passariam a ser compreensivas também para as máquinas (BERNERS-LEE; HENDLER; LASSILA, 2001).

Paralelamente, o desenvolvimento de sistemas passou a adotar técnicas baseadas na orientação a serviços, que possibilitam o reuso de software e aceleram o desenvolvimento de novos sistemas. Destaca-se como principal tecnologia empregada para essa finalidade a arquitetura de Web Services.

Os Web Services passaram a não ser somente reutilizados, mas também a ser combinados para compor novas funcionalidades. Chamamos essa técnica de composição de serviços. A composições são feitas pelos desenvolvedores, já que as informações contidas nos Web Services não possuem descrição semântica.

Os Serviços Web Semânticos adicionam a semântica definida por Berners-Lee nos Web Services, tornando-os compreensíveis para máquinas. Isso possibilita a automação da composição de serviços, já que torna-se possível identificar o significado de operações, entradas e saídas.

1.1 MOTIVAÇÃO

As abordagens encontradas na literatura para composição de serviços podem ser agrupadas em três categorias, conforme o grau de automação das composições: composições automáticas, composições semiautomáticas, e composições manuais.

Composições manuais levam em conta apenas o conhecimento semântico que o usuário possui sobre os serviços. Geralmente são feitas por programadores na fase de desenvolvimento dos sistemas. Qualquer alteração em um serviço da composição acarreta em mudanças no código para adaptá-lo.

Já as composições automáticas podem ser feitas em tempo de execução e envolver usuários com menos conhecimento técnico, já que é necessário apenas descrever as entradas e a saída esperada. Porém, o problema de composições automáticas está no tempo para executar o processo, que inclui o processo de requisição, descoberta, e mecanismo de composição de serviços (LAGA; BERTIN; CRESPI, 2009). No

trabalho de Hobold (2012) ficou evidente o alto custo computacional do processo de composição automática.

Laga, Bertin e Crespi (2009) apontam que o maior envolvimento do usuário diminui consideravelmente o tempo para construção das composições. Isso é interessante para abordagens que queiram aproveitar o melhor que a semântica traz para a automação de serviços, sem diminuir a participação do usuário.

1.2 PERGUNTA DA PESQUISA

Este trabalho visa responder a seguinte pergunta de pesquisa: **É possível a criação de um mecanismo de sugestão de serviços, levando em conta características semânticas das entradas e saídas das operações e a disponibilidade de seus serviços?**

1.3 OBJETIVOS

1.3.1 Objetivo Geral

Esse trabalho visa especificar uma abordagem para posterior implementação de um protótipo capaz de sugerir dinamicamente operações que sejam semanticamente compatíveis com a composição sendo criada.

1.3.2 Objetivos Específicos

A partir do objetivo geral descrito, é possível dividi-lo em três objetivos específicos:

1) Definir um algoritmo para a seleção de serviços, considerando a semântica das entradas e saídas das operações e critérios de disponibilidade.
2) Verificar a aplicabilidade da proposta por meio do desenvolvimento de um protótipo.
3) Criar ou adaptar uma interface gráfica para interação com o usuário.
4) Avaliar, através de testes de desempenho, o comportamento da abordagem proposta com um grande número de serviços.
5) Avaliar através de simulações o comportamento do algoritmo criado em cenários específicos.

1.4 MÉTODO DA PESQUISA

Os métodos utilizados para atingir o objetivo do trabalho foram:

1) Estudo sobre tecnologias da Web Semântica;
2) Estudo sobre tecnologias de Serviços Web Semânticos;
3) Estudo sobre composição de serviços;
4) Definição de um arquitetura capaz de sugerir serviços para uma composição de serviços;
5) Criação de uma base de dados contendo Serviços Web Semânticos;
6) Desenvolvimento de um protótipo que permita a validação da abordagem proposta;
7) Avaliar a proposta através da realização de testes de performance e simulações.

1.5 ORGANIZAÇÃO DO TRABALHO

Esse trabalho está organizado em três capítulos. No Capítulo 2 são apresentados os principais conceitos relacionado com os Serviços Web Semânticos, incluindo ontologias e tecnologias existentes para representação de Serviços Web Semânticos; é realizada uma revisão bibliográfica sobre composições de serviços; e, por fim, são revisados os principais trabalhos relacionados a essa proposta e é apresentada a ferramenta de workflow que será estendida neste trabalho.

O Capítulo 3 apresenta a arquitetura da proposta para composição semiautomática de serviços em uma ferramenta de workflow, através de sugestões baseadas no *matching* semântico entre os serviços que estão sendo compostos.

Em seguida o Capítulo 4 apresenta detalhes da implementação da proposta, a tecnologias utilizadas, a integração da proposta com uma ferramenta de composição de serviços, avaliação da proposta, e por fim a comparação com os trabalhos relacionados.

Por fim, o Capítulo 5 encerra os trabalhos com as conclusões, contribuições, possíveis trabalhos futuros e a publicação que o trabalho gerou.

2 FUNDAMENTOS TECNOLÓGICOS

2.1 WEB SEMÂNTICA

As primeiras páginas Web foram originalmente desenvolvidas para permitir o compartilhamento de informações através da rede entre pessoas e grupos de trabalhos dispersos fisicamente. Composto em grande parte por páginas HTML, essa fatia do que atualmente é a Web pode ser chamada de a Web Visual. A maioria das informações dispostas nessas páginas são apenas para consumos de humanos, já que o seu verdadeiro significado não é apresentado de maneira que computadores possam interpretar (CARDOSO, 2007).

Berners-Lee, Hendler e Lassila (2001) apresentam o conceito de uma Web onde as máquinas seriam capazes de compreender o significado de um conteúdo de uma página Web. Chamada de Web Semântica, a mesma seria um extensão da Web atual.

A Web Semântica tem dois objetivos principais (HAWKE et al., 2013):

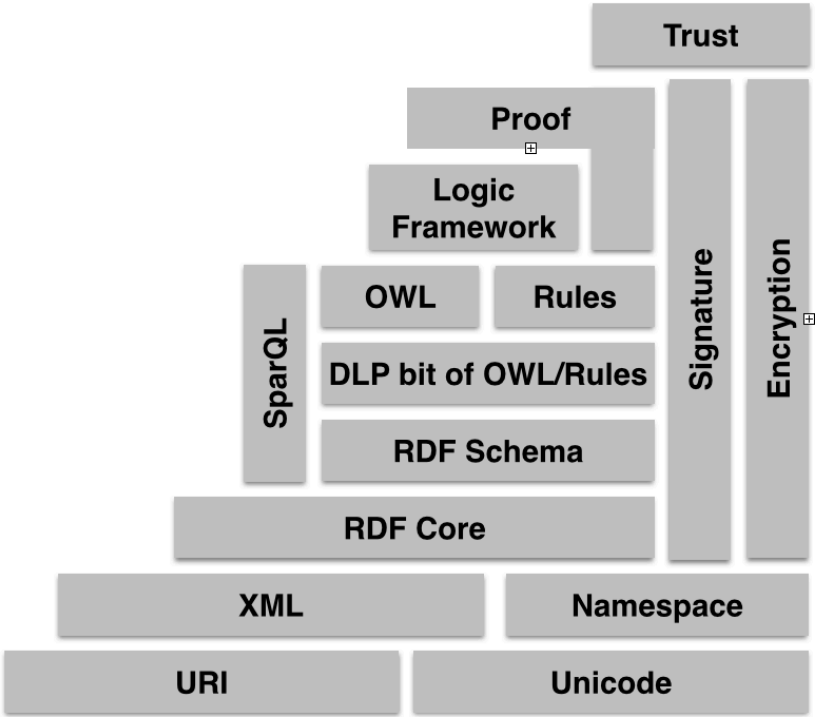
- Permitir a integração e combinação de dados retirados de fontes diferentes empregando um formato comum de representação de dados;
- registrar como o dado se relaciona a um objeto do mundo real empregando uma linguagem capaz de estabelecer relacionamentos entre dados.

A Web semântica é composta por um conjunto de padrões, ferramentas e tecnologias que geralmente é apresentado na forma de uma pilha de camadas, conforme ilustrado na Figura 1.

Na base desta arquitetura encontram-se o URI (*Universal Resource Identifier*) e o Unicode. URI é um conjunto de caracteres ASCII formatados com a intenção de identificar recursos físicos ou abstratos da Web Semântica. Já o Unicode é um padrão de codificação de caracteres, onde cada caractere possui um único código numérico associado a si (CARDOSO, 2007).

As camadas superiores de XML, RDF, e de Ontologias e OWL serão tratadas nas próximas seções. Já as outras camadas não farão parte deste trabalho, dessa forma não serão descritas.

Figura 1 – Arquitetura em Camadas da Web Semântica (BERNERS-LEE, 2005)



2.1.1 XML

XML (*Extensible Markup Language*) é uma linguagem de marcação de propósito geral, desenvolvida para descrever a estrutura de documentos e também um padrão aceito para troca de mensagem pela Web. Utiliza o conceito de *tags* assim como o HTML, porém sem restrições para criação de novas *tags* (BIKAKIS et al., 2013). *Namespace*[1] é uma forma simples de evitar conflitos entre vocabulários diferentes, associando-os a referências URI.

O XML Schema é utilizado para definir, descrever e catalogar vocabulários XML para classes de documentos XML. Criando restrições e documentando o significado, utilização e relacionamento da partes constituintes: tipos de dados, elementos e seus conteúdos, atributos e

[1] http://www.w3.org/TR/xml-names

seus valores, entidades e seu conteúdo e notação (MALHOTRA; MALONEY, 1999).

Do ponto de vista de interoperabilidade semântica, XML possui restrições. Um aspecto significativo é que não é possível reconhecer semântica de um domínio particular, pois XML foca na estrutura de seus arquivos e não impõe nenhum meio comum para intepretação dos dados (CARDOSO, 2007).

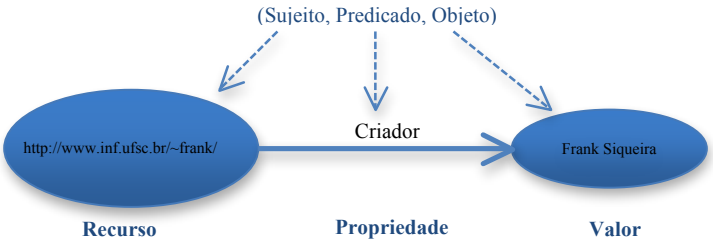
2.1.2 RDF

A linguagem RDF (*Resource Description Framework)* foi criada para padronizar a definição e o uso de metadados na Web que fossem compreensíveis por máquinas. Foi a primeira linguagem criada especialmente para a Web Semântica.

Trata-se de uma linguagem de propósito geral de metadados que representa informação na Web e fornece um modelo para descrever e criar relação entre recursos. Um recurso pode ser algo como uma pessoa, uma música ou uma página Web. Com RDF é possível predefinir primitivas de modelagem para expressar a semântica de um dado em um documento sem presumir como o mesmo está estruturado (CARDOSO, 2007). Seu modelo básico de dados é fundamentado no conceito de triplas: sujeito; predicado; e objeto. O sujeito identifica um recurso, o predicado suas propriedades e, por fim, o objeto é o valor associado a esta tripla, podendo ser o sujeito de outra tripla (BIKAKIS et al., 2013).

A Figura 2 apresenta graficamente o conceito de tripla e a Figura 3 mostra seu equivalente em RDF. O exemplo apresenta o sujeito como sendo a URI *http://www.inf.ufsc.br/~frank* que tem como predicado o criador do sujeito que tem o valor "Frank Siqueira".

Figura 2 – Representação gráfica de um tripla em RDF (CARDOSO, 2007).



Complementarmente ao RDF, o RDF Schema (RDFS) possibilita a definição de vocabulários para RDF através de uma linguagem de ontologia. Ao contrário de um XML Schema, que descreve apenas a sintaxe de um documento XML, o RDFS é um vocabulário para definir classes, propriedades, hierarquia e restrições (FENSEL et al., 2007). Comparado ao RDF, o RDFS permite a construção de um modelo de objeto onde é possível definir recursos entre classes, propriedades e valores (CARDOSO, 2007).

Figura 3 – Exemplo de um arquivo RDF (CARDOSO, 2007).

```
<?xml version="1.0" ?>
<RDF xmlns = "http://w3.org/TR/1999/PR-rdf-syntax-19990105#"
       xmlns:DC = "http://dublincore.org/2003/03/24/dces#">
    <Description about = "http://http://www.inf.ufsc.br/~frank/">
        <DC:Creator> Frank Siqueira </DC:Creator>
    </Description>
</RDF>
```

2.1.3 Ontologias

Segundo Fensel et al. (2007), uma ontologia é uma especificação formal e explícita de um conhecimento compartilhado. Proveniente da filosofia, seus primeiros usos em ciência da computação foram na área de inteligência artificial, proporcionando um comum entendimento de um domínio particular (CARDOSO, 2007).

Ontologias podem ser utilizadas para: auxiliar a comunicação entre humanos; permitir a interoperabilidade entre aplicações; e melhorar a qualidade de sistemas (JASPER; USCHOLD, 1999). Através de terminologias consensuais, formalidade e semântica do "mundo real" as ontologias criam um vocabulário padronizado que fornece um conjunto de construtores para a criação de conhecimento significativo de maneira bem definida e inequívoca. Normalmente são representadas em uma linguagem baseada em lógica, para que possam ser feitas distinções detalhadas entre classes, propriedades e relações (CARDOSO, 2007).

2.1.3.1 OWL

Existem várias linguagens para representações de ontologias, sendo a OWL (*Web Ontology Language*) o padrão mais utilizado. Trata-se de uma linguagem de marcação semântica para compartilhamento de ontologias na Web (BOUZID; CAUVET; PINATON, 2012). Desde

2004 a OWL é uma recomendação da W3C e é considerada uma das tecnologias padrão para o desenvolvimento da Web Semântica (OWL WORKING GROUP).

Sua origem está vinculada à iniciativa DAML+OIL, desenvolvida por um grupo de pesquisadores europeus e americanos por volta do ano 2000, culminando posteriormente na linguagem OWL (KLUSCH, 2008). Encontra-se no nível de vocabulários ontológicos e complementa as linguagens RDF e RDFS, fornecendo um vocabulário mais expressivo junto com formalismo baseado em descrições lógicas (KOPECKY et al., 2007). Uma ontologia em um arquivo OWL pode ser descrita como:

- Anotação: representa um metadado;
- Axioma: especifica características de classes e propriedades;
- Fatos: dados sobre um indivíduo ou parte de um identificador de indivíduo.

OWL acompanha três sublinguagens com expressividade diferentes para o uso de comunidades específicas de usuários (MCGUINNESS; HARMELEN, 2004):

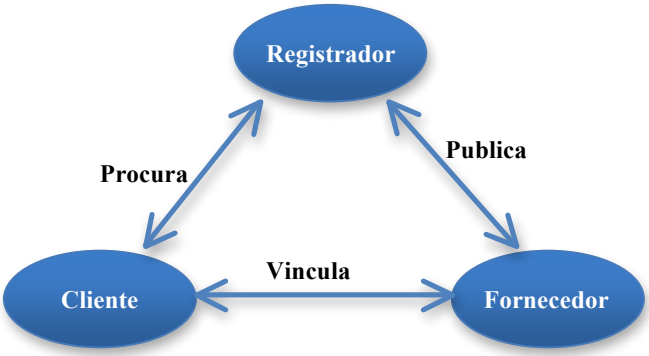
- OWL Lite: indicada para usuários que necessitam apenas de classificações hierárquicas e simples restrições;
- OWL DL: permite ao usuário utilizar o máximo de expressividade enquanto mantém completude e decidibilidade computacional.
- OWL Full: recomendada para usuários que querem o máximo de expressividade e liberdade sintática do RDF sem garantias computacionais. É improvável que algum software de raciocínio seja capaz de cobrir todas as funcionalidade do OWL Full.

Em 2009 a W3C apresentou uma nova versão da OWL, a OWL2 (OWL 2 *Web Ontology Language*). A nova versão possui uma estrutura semelhante e completa compatibilidade com a versão anterior, apresentando novos recursos, comumente chamados de "açúcar sintático", que não mudam a expressividade ou a semântica da versão anterior (BIKAKIS et al., 2013).

2.2 WEB SERVICES

A tecnologia de Web Services baseia-se na arquitetura Orientada a Serviços (SOA). Nessa arquitetura, um serviço nada mais é do que a representação de uma funcionalidade de negócio que aceita uma ou mais requisições e retorna uma ou mais respostas através de uma interface bem definida (JOSUTTIS, 2008).

Figura 4 – Modelo da arquitetura SOA (KREGER, 2001)



Segundo Kreger (2001) a arquitetura SOA é baseada no modelo apresentado na Figura 4. Nesse modelo as interações são realizadas entre três papeis: o fornecedor, o cliente, e registrador de serviços. As interações desse modelo envolvem as operações de publicação, procura e vinculação. Em um cenário típico, o fornecedor do serviço define a descrição do serviço e a publica para o cliente ou para o registrador de serviço. O cliente, por sua vez, utiliza uma operação de busca para recuperar a descrição do serviço e utiliza a descrição para estabelecer um vinculo com o fornecedor e invocar a implementação do serviço.

Por sua vez, a tecnologia de Web Services adota padrões que permitem que seus serviços sejam interoperáveis entre diferentes aplicações, rodando em um ambiente de rede contendo uma variedade de plataformas e frameworks (BOOTH, 2004).

A tecnologia de Web Services envolve várias camadas de padrões inter-relacionados, entre as quais destacam-se XML, SOAP e WSDL.

Como visto anteriormente, XML fornece um padrão para o formato de dados. Similarmente a tecnologias da Web Semântica, outros níveis da tecnologia de Web Services baseiam-se em XML.

SOAP é um protocolo para troca de mensagens em um ambiente descentralizado e distribuído. Consiste de três partes: um envelope que

define o que está na mensagem e como processá-la; um conjunto de regras de codificação de instâncias de tipos de dados definidos pela aplicação; e, por último, uma convenção de como representar chamadas e respostas de funções remotas (ZHOU; KOIVISTO; NIEMELÄ, 2006).

Por fim, WSDL (*Web Services Description Language*) é uma linguagem para descrição de serviços, que permite especificar as operações suportadas pelo serviço e o formato da mensagem para comunicar-se com ele. Na descrição de cada operação são especificadas suas entradas e saídas (MALAIMALAVATHANI; GOWRI, 2013).

No modelo da Figura 4, as interações de publicação e procura são realizadas através de WSDL e o vínculo entre cliente e fornecedor ocorre por meio da troca de mensagens SOAP.

2.3 WEB SERVICES SEMÂNTICOS

Conforme visto anteriormente, a Web, antes da definição de Web Semântica por Berners-Lee, Hendler e Lassila (2001), era apenas visual. As informações contidas em sua páginas eram inteligíveis apenas para humanos; já para máquinas, as páginas continham apenas conteúdo sintático. Após isso foi possível, através das tecnologias descritas, como RDF e OWL, adicionar significado ao conteúdo existente na Web, originando a Web Semântica.

O conceito de Web Services têm como objetivo possibilitar a integração de aplicações através de um *framework* padronizado para a extração de informação dinamicamente pelas aplicações. Isso contrasta com uma página Web padrão, na qual o conteúdo da página HTML é estaticamente obtido.

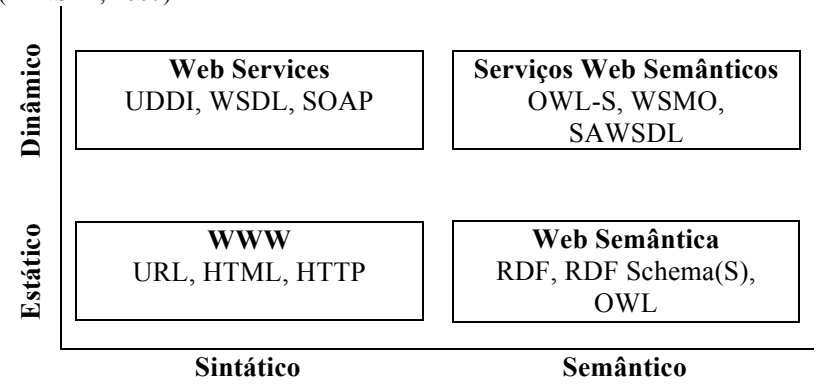
Porém os padrões atuais de Web Services (WSDL, SOAP, UDDI) foram projetados para representar apenas a capacidade e os requisitos de um serviço. A falta de capacidade de representar semântica em um Web Service faz com não seja possível a criação de integração automatizada de serviços (CARDOSO, 2007).

A maneira de tornar possível a automatização da utilização de serviços como descoberta, seleção e composição é fazer com que os serviços em si sejam processáveis por máquinas, assim como visto em Web Semântica (FENSER et al., 2007). A convergência entre a Web Semântica e Web Services resulta nos Serviços Web Semânticos (SWS) (SCHUMACHER; SCHULDT; HELIN, 2008).

A Figura 5 apresenta a correlação entre as tecnologias descritas anteriormente em função de suas características estáticas ou dinâmicas, e sintática ou semântica. No topo, à direita, encontram-se os Serviços

Web Semânticos, que aliam a dinamicidade dos Web Services com a semântica da Web Semântica.

Figura 5 – Relação dos Serviços Web Semânticos com as demais tecnologias (FENSEL, 2007)



Assim, os Serviços Web semântico permitem especificação mais compreensiva de vários aspectos de um serviço, dessa forma suportando (MARTIN; DOMINGUE, 2007):

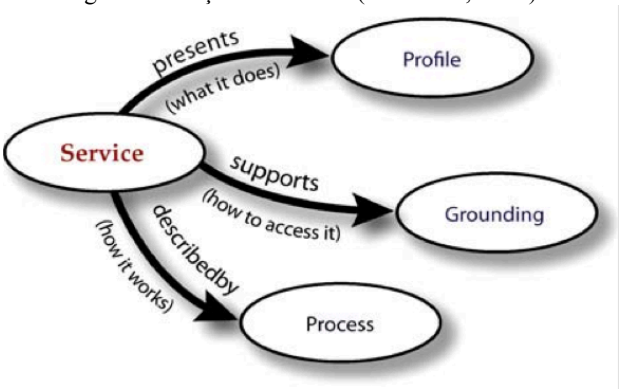
- Maior automação da seleção e invocação de serviços;
- Tradução automática do conteúdo de mensagem trocadas entre serviços heterogêneos;
- Abordagem automática ou semiautomática de composição de serviços;
- Uma abordagem mais compreensiva para o monitoramento e recuperação do serviço em casa de falha.

A semântica faz com que serviços que possuam diferentes descrições WSDL, em outras palavras, serviços que sejam sintaticamente diferentes, mas que ofereçam a mesma funcionalidade, sejam vistos como serviços equivalentes (DUSTDAR; SCHEREINER, 2005).

2.3.1 OWL-S

OWL-S é uma ontologia de alto nível utilizada para descrever a semântica de serviços com base no padrão OWL e fundamento em WSDL. A linguagem OWL-S é primariamente formada por três subontologias que precisam fornecer tipos de conhecimento sobre o serviço, tais como: o que faz; como acessá-lo; e como funciona. A Figura 6 apresenta uma representação gráfica da estrutura da ontologia do serviço em OWL-S.

Figura 6 – Ontologia do serviço em OWL-S (MARTIN, 2007)



A subontologia do *Service Profile*, ou perfil do serviço, fornece um conjunto de conceitos para descrever explicitamente as capacidade de um serviço, com o objetivo de suportar o descobrimento do serviço. Para o fornecedor do serviço permite informar o que o serviço faz, e possibilita que o cliente comunique o que espera do serviço que está utilizando. Isso evita que as descrições sejam extraídas de propriedades como nome do serviço ou da companhia que o fornece (MARTIN, 2007).

Service Process Model descreve a composição de um ou mais serviços, que é a promulgação controlada dos processos constituintes junto com o respectivo padrão de comunicação. Um processo em OWL-S pode ser atômico, simples ou composto (SCHUMACHER; SCHULDT; HELIN, 2008). Na especificação oficial do OWL-S a semântica do *Service Process Model* não foi definida, porém existem diversas abordagens que podem ser utilizadas para formalizar a semântica da orquestração. Uma delas é o BPEL[2] (*Business Process Execution Language*) (KLUSCH; FRIES; SYCARA, 2009).

[2] http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html

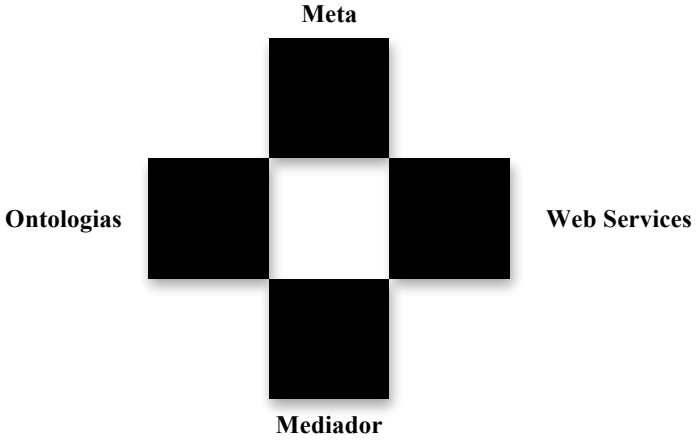
A base (*Service Grouding*) de uma descrição de serviço OWL-S fornece um ligação pragmática entre as definições lógicas e baseadas em XML de um serviço com o propósito de facilitar a sua execução (SCHUMACHER; SCHULDT; HELIN, 2008).

2.3.2 WSMO

O WSMO (*Web Service Modeling Ontology*) compartilha com OWL-S a mesma visão de que ontologias são essenciais para suportar descoberta, interoperação e composição automática de Web Services. A maneira de declarar entradas, saídas, precondições e resultados, associando-as com o serviço, é semelhante à forma utilizada pelo OWL-S, embora suas terminologias sejam um pouco diferentes. Porém, diferentemente de OWL-S, WSMO não provê um notação para construir processos compostos em termos de fluxos de dados (MARTIN, 2007).

Através de especificações ontológicas para os elementos principais dos Serviços Web Semânticos, WSMO é composto principalmente pelos quatro elementos apresentado na Figura 7.

Figura 7 – Elementos que compõem o WSMO (FENSEL, 2007)



O elemento Meta descreve aspectos relacionados ao que o cliente espera do serviços sendo buscado. As ontologias fornecem a formalização lógica das informações utilizadas por todos os componentes da modelagem. Os Web Services são as entidades que fornecem algum valor em determinado domínio. Por fim, o elemento

Mediador resolve eventuais problemas de interoperabilidade entre serviços ou ontologias nos níveis de dado, protocolo e processo (SCHUMACHER; SCHULDT; HELIN, 2008).

2.3.3 SAWSDL

SAWSDL (*Semantic Annotations for WSDL and XML Schema*) é uma recomendação W3C para Serviços Web Semânticos. Foi concebido pelo grupo de trabalho em anotações semânticas para WSDL, que iniciou seu trabalho em 2006 após a constatação de falta de concordância sobre o que um Serviço Web Semântico deveria fazer (KOPECKY et al., 2007).

Diferentemente da linguagem OWL-S e WSMO, SAWSDL não especifica uma nova linguagem ou uma ontologia de alto nível para a descrição de serviços semânticos, mas permite mecanismos para que conceitos ontológicos definidos fora de um documento WSDL possam ser referenciados semanticamente por anotações inseridas em elementos descritivos do WSDL. Seus princípios fundamentais de projeto são que: a especificação permite a criação e utilização de anotações semânticas em Web Services pela capacidade de extensão do WSDL; é indiferente à linguagem de representação semântica utilizada; permite a utilização de anotação não só pra descoberta de Web Services, mas também para a invocação dos mesmos (SCHUMACHER; SCHULDT; HELIN, 2008);

SAWSDL introduz três novos atributos de extensão para utilizar nos documentos WSDL e XML Schema: *modelReference, liftingSchemaMapping e loweringSchemaMapping*. O *modelReference* define a associação entre um componente WSDL ou XML Schema e um conceito dentro de um modelo semântico. Pode ser utilizado para anotar interfaces, operações, e falhas em um WSDL e em tipos de definições, e declarações de atributos e elementos em um XML Schema (FARRELL; LAUSEN, 2007).

Os outros dois atributos de extensão, *liftingSchemaMapping* e *loweringSchemaMapping* são utilizados no XML Schema para declarações de elementos, definições de tipos complexos e simples. Ambos permitem criar mapeamentos entre dados semânticos em domínios referenciados pelo *modelReference* e XML, que podem ser utilizados durante a invocação de serviços (SCHUMACHER; SCHULDT; HELIN, 2008).

2.4 COMPOSIÇÃO DE SERVIÇOS

A composição de serviços é o processo de combinar e vincular serviços, atômicos ou compostos, com o intuito de criar um novo serviço (KAPITSAKI et al., 2007). Ao compor um serviço, a lógica do cliente é implementada por outros serviços. O processo é análogo à criação de workflows, onde a lógica da aplicação é feita através da composição de aplicações autônomas. Um cliente que invoca um serviço composto pode tornar-se um serviço (DUSTDAR; SCHREINER, 2005).

Autores classificam de diferentes formas as composição de serviços (KAPITSAKI et al., 2007). Neste trabalho adotaremos a classificação de Dustdar e Schereiner (2005), complementando-a com o trabalho de Laga, Bertin e Crespi (2009).

As estratégias para composição de serviços são classificadas de cinco formas: quanto a tempo que ocorrem; dirigidas por modelo; declarativas; grau de automação; baseada no contexto (DUSTDAR; SCHEREINER, 2005).

Composições estáticas ou dinâmicas concernem ao tempo que as composições ocorrem. As composições estáticas originam-se durante a fase de modelagem, quando a arquitetura e a modelagem do software são planejadas. Então os serviços que farão parte da composição são escolhidos, vinculados e implantados. Essa abordagem funciona enquanto o ambiente ou o próprio serviço continuarem iguais. Novos serviços podem ocasionar inconsistência no sistema, gerando mudanças na arquitetura (DUSTDAR; SCHEREINER, 2005). Já as composições dinâmicas são feitas em tempo de requisição, permitindo que a descoberta e criação da composição de serviços não fiquem amarradas ao desenvolvedor. Em caso de problema ou alteração de um serviço, a composição dinâmica adapta-se e substitui o Web Service por um serviço similar (HOBOLD, 2012).

A composição de serviços dirigida por modelo facilita o desenvolvimento e gerenciamento de composições dinâmicas de serviço. Utiliza regras de negócio para estruturar e agendar a composição de serviços, e descrever a seleção e ligação de serviços (DUSTDAR; SCHEREINER, 2005).

Na estratégia de composição declarativa de serviços as requisições do cliente são expressas utilizando uma linguagem formal. O processo de composição consiste de duas fases: a primeira fase pega uma situação inicial e o objetivo desejado, e constrói planos genéricos para atingir esse objetivo. Posteriormente um dos planos é escolhido, descobre-se serviços apropriados e constrói-se um workflow a partir dele (DUSTDAR; SCHEREINER, 2005).

As estratégias de composições manual ou automática são recorrentes em artigos sobre composição de serviços (DUSTDAR; SCHREINER, 2005; SYU et al., 2012; KAPITSAKI, et al. 2007). Uma composição manual é aquela realizada inteiramente por intermédio do desenvolvedor, que possui conhecimento dos serviços existentes e os invoca a fim de obter o serviço composto. Essa etapa é realizada antes do serviço ser consumido pelo usuário final (HOBOLD, 2012). Já uma composição automatizada é o processo de selecionar, combinar, integrar e executar automaticamente uma composição de serviços (MARTIN et al., 2007). Porém, mesmo uma abordagem totalmente automatizada ainda necessita da intervenção humana para formular a requisição da composição, geralmente contendo a especificação da entrada, saída desejada e possivelmente algum critério de otimização (SYU et al., 2012). Complementarmente, existe a abordagem semiautomática para composição de serviços. Nessa abordagem a composição do serviço é realizada pelo usuário final através de uma interface gráfica ou linguagem formal (LAGA; BERTIN; CRESPI, 2009). A interface gráfica indica os Web Services disponíveis, e conforme o usuário vai criando a composição, a ferramenta apresenta serviços passiveis de inclusão, podendo haver semântica por trás da sugestão (HOBOLD, 2012).

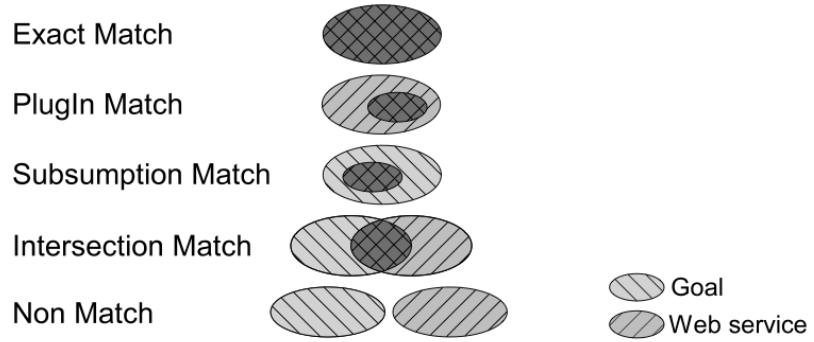
Syu et al. (2012) destacam que apesar de existirem vários ramos de pesquisa para automatizar a composição de serviços, existem três aspectos centrais relacionados. São eles: classificação, combinação, e a seleção de serviços. Há ainda dois aspectos transversais, que afetam diretamente os outros: descrição e *matching* do serviço. Como esses dois últimos aspectos influenciam os demais, serão definidos primeiro.

Do ponto de vista de SOA, a descrição de um serviço é importantíssima, pois a reutilização dos serviços é similar ao conceito de caixa preta, segundo o qual o usuário não tem conhecimento de como algo é realizado. A única maneira de sabê-lo é através de sua descrição. O padrão mais aceito para Web Services é o WSDL, porém o poder descritivo de um WSDL é suficiente apenas para construir interfaces sintáticas. Apenas descrição sintática não permite a criação de composições de forma automatizada, tornando necessário vincular semântica aos serviços. Por essa razão, a comunidade criou maneiras de ter ontologias associadas à descrição de serviços (OWL-S, WSMO, SAWSDL) (SYU et al., 2012).

O *Matching* estabelece correspondência, através de inferência, entre objetivos especificados na requisição e os Web Services disponíveis, a fim de descobrir aqueles com potencial de realizar o que

foi especificado (HOBOLD, 2012). Fensel et al. (2007) identificam cinco noções de *matching*, sendo que cada noção denota um relacionamento lógico diferente que deve ser verdadeiro caso um Web Service seja adequado para atingir o objetivo funcional especificado. Um *Exact Match* é verdadeiro se e somente se todas as possíveis instâncias da ontologia que satisfazem o Web Service também satisfaçam os objetivos, e também não exista qualquer instância que satisfaça apenas o Web Service ou o Objetivo. Em contraste, um *Intersection Match* ocorre caso exista uma possível instância da ontologia que satisfaça tanto o objetivo quanto o Web Service. A noção de *PlugIn Match* ocorre nos casos que a instância da ontologia do objetivo especificado é uma subclasse da instância da ontologia do Web Service. Já o *Subsumption Match* é quando a instância da ontologia do Web Service é uma subclasse da instância do objetivo especificado, o oposto do *PlugIn Match* (PRAZERES, 2009). Por fim, *Non Match*, ocorre quando não há correspondência semântica entre o Web Service e o objetivo especificado. A Figura 8 sumariza a noção de *matching*.

Figura 8 – Tipos de *Matching* (FENSEL, 2007)



Syu et al. (2012) distinguem a Classificação de serviços dos demais aspectos centrais, pois consideram que a classificação não lida diretamente com a composição de serviços. Seu papel é auxiliar as etapas de combinação e seleção. Em algum casos, melhora o desempenho e a eficiência das composições através da diminuição do número de serviços. Uma maneira de obter essa melhora no desempenho é considerar serviços com a mesma funcionalidade como um único serviço. Para a seleção de serviços, a classificação pode agregar os serviços que possuem os mesmos requisitos funcionais e que diferem

quanto a serviços não funcionais para que posteriormente o seletor decida o serviço mais apropriado entre eles.

A Combinação de serviços é vista como o aspecto mais importante da composição de serviços, funcionando como o esqueleto do serviço composto para atender aos requisitos funcionais do requisitante. Embora cada solução e detalhe de implementação seja diferente, grosseiramente existem dois mecanismos de combinação. O primeiro cria um *template* abstrato da combinação, que depois é encaminhado para a abordagem que realiza a seleção. Já o segundo mecanismo vincula os serviços disponíveis e conhecidos em um serviço composto.

E a fase de seleção de serviços é responsável por otimizar a composição. Trata-se de uma atividade pós-planejamento em que os melhores serviços são selecionados através de parâmetros funcionais (Entrada, Saída, Pré-condições, e Efeito), podendo-se levar em consideração propriedades não funcionais dos mesmos (como por exemplo QoS, custo do serviço, disponibilidade e localização geográfica) (PRAZERES, 2009).

2.5 TRABALHOS RELACIONADOS

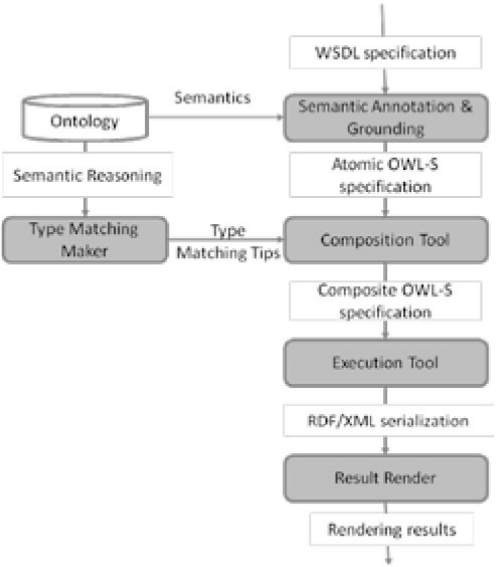
O objetivo desta seção é apresentar trabalhos que utilizem Serviços Web Semânticos para criar serviços compostos.

O trabalho de Pi et al. (2012) propõe um framework para composição semiautomática de serviços web semânticos através de uma interface gráfica, o Flow Editor. O framework possibilita ao usuário converter Web Services comuns em serviços web semânticos representados em OWL-S. Para isso o usuário submete o endereço do WSDL, faz as anotações semânticas das entradas e saídas e a ferramenta gera o OWL-S do serviço. A ferramenta também é capaz de executar e exportar as composições criadas. A Figura 9 apresenta as características do Flow Editor.

Os módulos do Flow Editor encontram-se destacados. O Módulo de Anotação e Fundamentação Semântica é o módulo responsável por converter os Web Services em Serviços Web Semânticos. O usuário especifica o endereço de um arquivo WSDL e adiciona a semântica por trás das entradas e saídas do serviço a partir de uma base contendo as ontologias cadastradas. Em seguida é gerado o serviço em OWL-S que poderá ser utilizado pela ferramenta gráfica para a composição. O módulo de validação de *matching* verifica se dois serviços podem ser ligados através da análise, feita por um raciocinador semântico, dos

conceitos de suas entradas e saídas. Os dois últimos módulos são responsáveis por executar a composição e apresentar seus resultados em XML ou outro formato que possa ser utilizado por outras aplicações.

Figura 9 – Modelo da arquitetura do FlowEditor



Meiyu e Maoji (2013) apresentam um esquema abstrato para composição de serviços web semânticos com base em requisitos dos usuários. No framework proposto existem dois módulos: o módulo de gerenciamento de serviços, e o módulo de gerenciamento de composição.

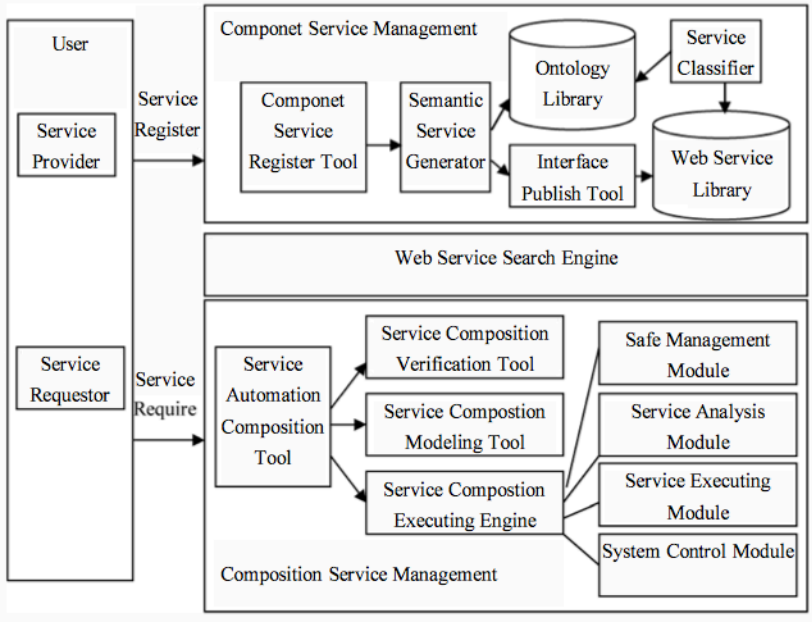
O módulo de gerenciamento de serviços é responsável pelo cadastramento de novos Serviços Web Semânticos. O componente responsável pelo cadastramento de novos serviços aceita serviços em WSDL ou descritos em OWL-S. Caso o serviço cadastrado seja descrito em WSDL há um componente responsável por gerar anotações semânticas. O módulo possui duas bibliotecas: uma para o cadastro dos Web Services, e outra para o cadastro das Ontologias.

Para buscar esses Web Services cadastrados existe um componente intermediário, o motor de buscas de Web Services. Esse componente combina a biblioteca de ontologias, e a de serviços, aos requisitos de usuários, que são enviados pelo componente contido no

módulo de composição, e a partir disso ele busca serviços para as necessidades dos usuários.

O Componente responsável pela automação da composição de serviços utiliza teoria de grafos para selecionar os melhores esquemas de composição. Para gerar o caminho, é realizada uma pesquisa reversa das saída esperadas até chegar à entrada. A Figura 10 representa o módulo de gerenciamento de serviços e o módulo de gerenciamento de composição.

Figura 10 – Modelo abstrato da arquitetura de composição semântica



Outro trabalho relacionado a composição semiautomática de serviços é o de Monteiro (2008). Através de uma interface gráfica em Java, o usuário é capaz de criar composições de acordo com descrições disponíveis em uma base de dados. O trabalho utiliza OWL-S para descrever os Serviços Web Semânticos. O autor separa a composição dos serviços em módulos de descoberta, composição e execução. A ferramenta possibilita criação de controles condicionais (*if*, *else*) e sequenciais (*loops*) na composição.

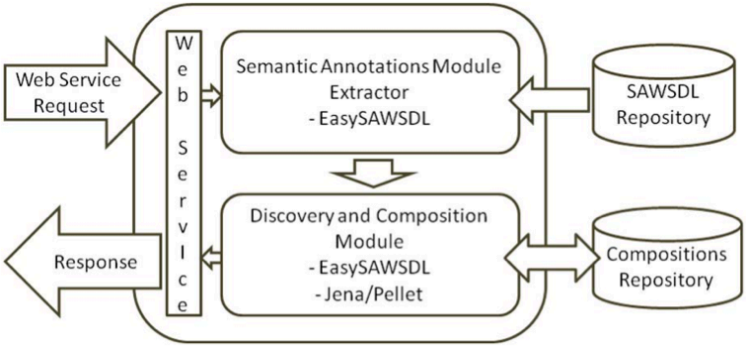
O trabalho de Hobold e Siqueira (2012) aborda a composição de serviço de maneira automática. Diferentemente dos trabalhos anteriores,

esse trabalho utiliza anotações em SAWSDL dentro dos elementos do WSDL para construir grafos da composição. O trabalho mira na automatização da descoberta e composição dos serviços. Para otimizar o processo de descoberta, na etapa de publicação extrai-se as anotações de suas operações, entradas, e saídas e as guarda em um banco de dados relacional.

Para requisitar um serviço ou uma composição, o usuário deve informar seis parâmetros, sendo eles: lista de entradas disponíveis; lista de saídas desejadas; lista de operações; profundidade máxima da composição; tempo de espera da busca; permitir reconstrução da composição. Os três primeiros parâmetros são URIs para conceitos definidos dentro de uma ontologia de domínio. Por ser uma abordagem que adota teoria dos grafos para composição, a profundidade máxima é o limite de serviços sequenciais que a composição pode ter. O tempo de espera é o tempo máximo que usuário deseja esperar para a composição ser encontrada. E o último parâmetro informa se composições previamente descobertas podem ser elegíveis. Em outras palavras, quando uma composição é descoberta, ela é armazenada para que futuramente possa se verificado se a composição satisfaz as necessidades de uma nova requisição.

A seleção de composições é feita baseando-se na qualidade semântica das mesmas. Se existir mais de um caminho que leva à mesma saída, o framework analisa qual possui o menor grau de divergência semântica. A descoberta e composição automática dos Serviços Web Semânticos é feita em tempo de execução. A Figura 11 mostra a arquitetura do SWSComposer. O repositório SAWSDL armazena as descrições das interfaces dos serviços, para que a etapa de descoberta ocorra mais rapidamente. Já o repositório de composição é o que armazena as composições descobertas anteriormente e é utilizado quando uma requisição permite a reutilização de composições. A arquitetura possui dois módulos: O módulo que extrai as anotações de novos serviços, e o módulo que realiza a descoberta e composição de serviços.

Figura 11 – Arquitetura do SWSComposer (HOBOLD; SIQUEIRA, 2012)



A Tabela 1 apresenta uma comparação entres os trabalhos correlatos pesquisados. Sumarizando informações como: tecnologia para descrição de serviços web semânticos; características da maneira que a composição é feita; e se levam em conta algum critério de qualidade de serviço em seu processo de composição.

Por ser tratar de um modelo abstrato para a composição de serviços, o trabalho de Meiyu e Maoji (2013) não foi incluído junto com os outros trabalhos na Tabela 1.

Tabela 1 – Comparação dos trabalhos relacionados

	Pi et al. (2012)	Monteiro (2008)	Hobold e Siqueira (2012)
Serviços Web Semânticos	OWL-S	OWL-S	SAWSDL
Forma da Composição	Semiautomática	Semiautomática	Automática
Momento da Composição	Requisição	Desenvolvimento	Requisição
Tipo da Composição	Estática	Estática	Dinâmica
Técnica da Composição	Interação com o usuário	Interação com o usuário	Grafos
Parâmetros de *QoS*	Não	Não	Não

3 O MECANISMO DE SUGESTÃO DE SERVIÇOS SEMÂNTICOS

O intuito deste trabalho é a criação de um mecanismo de sugestão de serviços web semânticos que auxilie o usuário na criação de composições de serviços. Esse mecanismo possibilitará a automação de certas etapas do processo de composição, como a verificação de compatibilidade entre entradas e saídas de operações, trazendo uma lista de operações ordenadas com base em critérios que determinam as operações mais indicadas para serem utilizadas na composição em construção.

Pelo fato de a abordagem não ser completamente automatizada, evita-se o *overhead* associado a processos que buscam criar composições completas, já que o foco da abordagem é atingir o *matching* entre a entrada da operação sendo sugerida e as saídas da composição criada.

O objetivo deste capítulo é apresentar a proposta desenvolvida. Primeiramente, apresentam-se as principais diferenças entre abordagens para composição de serviços com diferentes graus de automação. Em seguida, a arquitetura da proposta é exposta e descrita. Por fim, é feito um detalhamento de certos aspectos relevantes da arquitetura proposta.

3.1 DIVERGÊNCIAS ENTRE ABORDAGENS MANUAIS E AUTOMATIZADAS

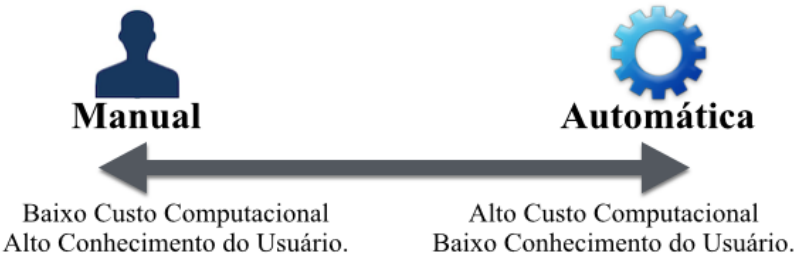
O processo inteiramente automatizado de composição de serviços está diretamente associado a uma ontologia bem descrita dos serviços que os integram, além de alto custo computacional para realizar a busca em profundidade de caminhos que satisfaçam a saída e entrada especificadas pelo usuário no início do processo. Esse aspecto é evidenciado pelo trabalho de Hobold e Siqueira (2012), onde a operação que ocupava o maior tempo de processamento consistia da busca em profundidade de composições que satisfizessem os parâmetros passados.

O caminho inverso, ou seja, a criação estritamente manual de composições, requer um vasto conhecimento técnico dos serviços que fazem parte do repositório. Isso restringe o número de usuários capazes de realizar com êxito a composição de serviços, já que o processo de inferência da semântica de entradas e saídas é feito exclusivamente pelo usuário. Já o custo computacional dessas operações é bastante reduzido,

havendo apenas a demora do usuário em interagir com a interface gráfica de uma ferramenta de composição.

A Figura 12 apresenta os dois lados de composições manuais e as composições totalmente automatizadas. Apesar de as composições manuais possuírem um baixo custo computacional, é pequena a quantidade de usuários capaz de compor serviços com sucesso. Já a abordagem totalmente automatizada amplia a gama de usuários capaz de utilizar a ferramenta, mas tem um alto custo computacional associado.

Figura 12 - Composições Manuais versus Composições Automáticas.

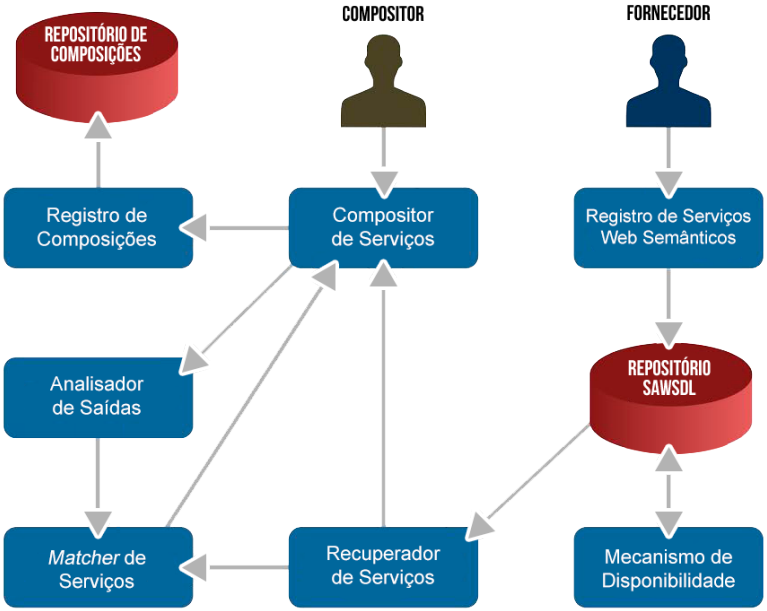


Laga, Bertin e Crespi (2009) afirmam que grande parte do gargalo do tempo de processamento das abordagens automatizadas pode ser minimizado através da diminuição do grau de automação, preferindo abordagens semi-automatizadas para a composição dos serviços. Dessa forma optou-se pela abordagem semi-automatizada nesse trabalho, diminuindo o conhecimento técnico exigido do usuário, sem adicionar o alto custo computacional de uma abordagem totalmente automatizada para a composição de serviços.

3.2 ARQUITETURA PROPOSTA

A proposta desse trabalho é de criar um mecanismo que utilize descrições semânticas de serviços Web para permitir um maior grau de automação no momento da composição de serviços. Essa automação tem como intuito auxiliar o usuário no momento da composição, sugerindo serviços com uma maior compatibilidade com a composição que está sendo construída.

Figura 13 – A Arquitetura do Mecanismo de Sugestão de Serviços Semânticos



O Mecanismo de Sugestão de Serviços Semânticos, de agora em diante chamado de S^{3}M (*Semantic Service Suggestion Mechanism*), seleciona operações de serviços que possuam entradas semanticamente compatíveis com as saídas das operações da composição sendo criada. Essa abordagem permite reduzir o conhecimento técnico do usuário que utiliza a ferramenta, pois ficará a cargo do mecanismo inferir se dois serviços possuem entradas e saídas compatíveis.

A Figura 13 apresenta a arquitetura do S^{3}M que fundamenta-se em dois perfis de usuário que interagirão com o sistema, dois repositórios para armazenamento de informações relativas a serviços e composições criadas e sete componentes de software. Os propósitos individuais dos componentes e suas inter-relações serão descritos nos parágrafos a seguir.

- O usuário Fornecedor é um perfil de usuário que possui conhecimento da localização de serviços semanticamente anotados.

- Já o usuário Compositor fica responsável pela criação de composições de serviços. O mesmo tem a necessidade de ter conhecimento técnico de como funciona a sintaxe dos serviços sendo utilizados para criar a composição, sendo necessário apenas um breve conhecimento do que cada serviço realiza.
- Agindo diretamente com o usuário Fornecedor, o componente de Registro de Serviços Web Semânticos serve como interface para o cadastro de serviços semanticamente anotados.
- Serviços são armazenados e persistidos na base de dados Repositório SAWSDL, onde os mesmos encontram-se disponíveis para o uso de outros componentes.
- O componente Mecanismo de Disponibilidade opera no *background* da arquitetura verificando a disponibilidade dos serviços persistidos no Repositório SAWSDL.
- O outro componente que interage diretamente com o repositório de serviços é o Recuperador de Serviços, que disponibiliza apenas os serviços que atinjam critérios satisfatórios de disponibilidade.
- O principal componente para o perfil de usuário Compositor é o elemento Compositor de Serviços. Esse componente provê uma interface gráfica responsável por apresentar os serviços para o usuário, providenciar um meio para que crie-se composições, e também apresentar os resultados do mecanismo de sugestão de serviços.
- O componente Analisador de Saídas opera com base nos serviços da composição sendo criada e obtém os parâmetros necessários para a execução do algoritmo de sugestão de serviços semânticos.
- O Mecanismo de Sugestão de Serviços é realizado pelo *Matcher* de Serviço, que recebe a lista de serviços existentes, junto com os parâmetros do Analisador de Saída, e gera um resultado que é passado para o componente Compositor de Serviço para que a escolha de serviços seja feita pelo usuário.
- Ao terminar de criar uma composição, é permitido ao usuário exportar os dados. Essa tarefa fica a cargo do componente Registro de Composição, que descreve a composição criada em um formato que posteriormente será armazenado.

- O último componente, o Repositório de Composição armazena essa composição em seu banco de dados.

Essa é uma breve descrição da arquitetura proposta. As seções subjacentes têm como objetivo detalhar o funcionamento dos componentes da arquitetura. As seções foram organizadas da seguinte maneira. Na seção 3.3 serão debatidos os componentes que armazenam os serviços ou interagem diretamente com o repositório de serviços. A seção 3.4 apresenta o funcionamento do componente de composição de serviços da arquitetura. A seção 3.5 dedica-se a esclarecer o funcionamento dos algoritmos por trás do mecanismo de sugestão. Por fim, a seção 3.6 relata como é feita a persistência das composições criadas.

3.3 ARMAZENAMENTO E INTERAÇÃO COM OS SERVIÇOS

Os componentes que interagem diretamente com o repositório de serviços foram agrupados nessa seção com o intuito de detalhar seus funcionamentos e como estes interagem com o repositório. Os componentes são: O Registro de Serviços Web Semânticos; Recuperador de Serviços; Mecanismo de Disponibilidade; e o componente responsável por armazenar os serviços, o Repositório SAWSDL;

O usuário Fornecedor interage exclusivamente com o Registro de Serviços Web Semânticos, que fornece uma interface através da qual são inseridos os serviços semanticamente anotados, que são processados e extraídos de maneira a otimizar o processo de comparação entre serviços. Outra função desempenhada por esse módulo é o de registrar em uma variável global o mínimo valor aceitável de disponibilidade, que é utilizada pelo módulo de *Matcher* de Serviços.

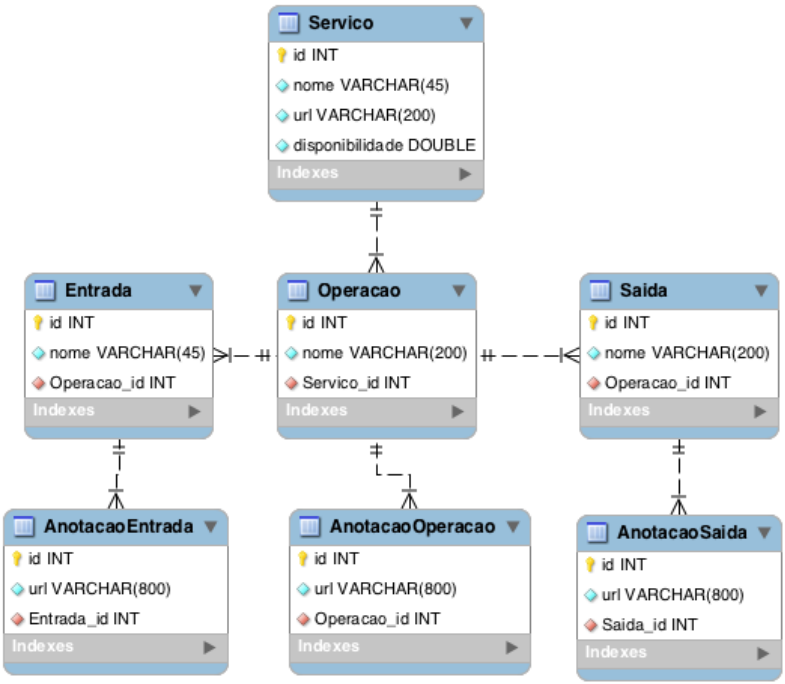
O processo de extração das informações de um serviço consiste na entrada da URL do serviço, e a extração das suas operações, entradas, saídas e as anotações semânticas associadas às mesmas.

Apesar de existirem tecnologias responsáveis pelo armazenamento dos serviços (por exemplo, UDDI) optou-se pela extração e armazenamento dos serviços de maneira similar ao da proposta de Hobold e Siqueira (2012) com o intuito de otimizar o desempenho do algoritmo de sugestão de serviços. Foi acrescentada à tabela de serviços uma coluna para registro da disponibilidade do serviço, que informa o quão disponível um serviço esteve nos últimos

ciclos de verificação. A disponibilidade de um serviço é monitorada periodicamente por outro componente.

A Figura 14 mostra o modelo entidade-relacionamento dos serviços armazenados no Repositório SAWSDL. A tabela de serviços, como dito anteriormente, armazena a URL do serviço e seu valor de disponibilidade, associado ao seu nome. Já as tabelas de operação, entrada, e saída armazenam seus respectivos nomes, e são associadas a tabelas que contém suas respectivas anotações semânticas.

Figura 14 – Modelo entidade relacionamento do repositório de serviços.



O módulo Mecanismo de Disponibilidade consiste de um procedimento que roda em background da arquitetura, verificando a disponibilidade de cada serviço existente no Repositório SAWSDL, e para cada um desses serviços é aplicado uma média ponderada que atualiza os valores de disponibilidade correspondentes. A média ponderada utilizada considera que a última verificação deve sempre ter mais impacto que as anteriores. Com base neste conceito, adota-se aqui o peso do último

dado de entrada como o peso equivalente à combinação de todas as entradas anteriores. A Equação 1 a seguir demonstra como o cálculo de disponibilidade é feito.

$v_n = 1$ *se o serviço estiver disponível;* 0 *caso o contrário.*
$dv_1 = v_1$
$dv_2 = \omega * v_2 + (1- \omega) * dv_1$
…
$dv_n = \omega * v_n + (1- \omega) * dv_{n-1}$ (1)

onde:
v_n: resultado da verificação de disponibilidade n ;
dv_n: métrica de disponibilidade para o intervalo n;
ω: peso da última verificação.

O intervalo de tempo no qual o serviço é monitorado e tem sua disponibilidade recalculada pode ser definido pelo usuário através de uma variável local.

O componente Recuperador de Serviços é o último componente do agrupamento de serviços que interagem diretamente com o repositório de serviços da composição. Seu papel é servir de intermediário entre o repositório e os componentes ligados diretamente a composição de serviços na arquitetura. Sua função primária é apresentar a lista de operações existentes para o módulo de composição, além de apresentar a mesma lista como parâmetro para o componente de sugestão de serviços semânticos. O componente opta por apresentar apenas as operações cujos serviços possuam um índice de disponibilidade superior ao estipulado pela variável global que registra o mínimo de disponibilidade admissível para um serviço. O pseudo-código a seguir ilustra seu funcionamento.

Algoritmo 1 – Listagem de operações que satisfaçam o critério de disponibilidade.

```
01 function retrieveOperations()
02   def operationList = ∅
03   for each service in SWADSL_Repository.services do
04     if service.availability >= GLOBAL.MIN_AVAILABILITY then
05        operationList.addAll(service.operations)
06     end if
07   end for
08   return operationList
09 end
```

Essa seção descreveu o usuário responsável por apresentar novos serviços para o S^3M, junto com os componentes de software que interagem diretamente com o repositório de serviços (Repositório SAWSDL). A próxima seção apresentará o papel do usuário Compositor e o componente Compositor de Serviços.

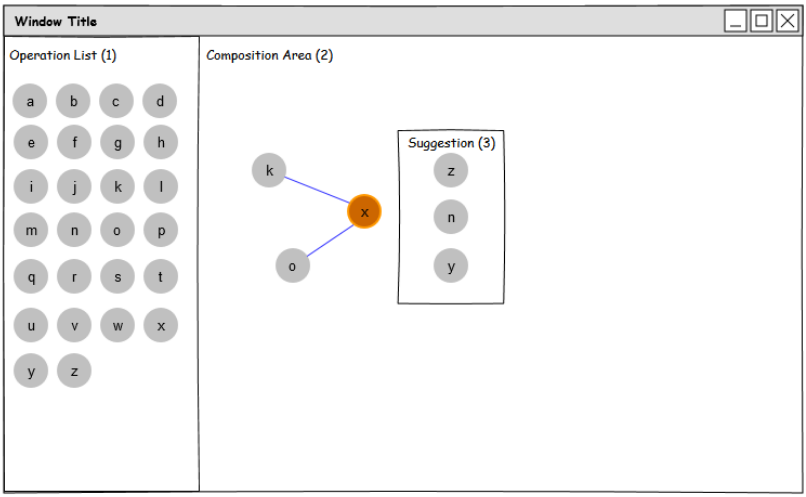
3.4 A COMPOSIÇÃO DE SERVIÇOS

Os componentes agrupados nessa seção são os componentes básicos de qualquer ferramenta de composição, seja ela de serviços ou ferramentas de *workflow*. No caso do S^3M, a interação entre usuário Compositor e a ferramenta de composição é realizada pelo componente Compositor de Serviços, que apresenta graficamente um meio de criar as composições, que são análogas à construção de grafos, onde as operações são os nós e as arestas representam a ligação entre as entradas e saídas das operações.

A diferença no caso da arquitetura apresentada é que, além das funcionalidades de uma ferramenta de composição comum, o S^3M apresenta também uma lista de operações semanticamente compatíveis com a atual composição. As sugestões são criadas dinamicamente, ou seja, em tempo de criação da composição, e são personalizadas conforme o nó selecionado pelo usuário no momento. Os serviços disponíveis são filtrados com base nos valores de disponibilidade dos serviços no repositório, porém o funcionamento do mecanismo de sugestão de operações é assunto para a próxima seção.

A Figura 15 exemplifica as três funcionalidades gráficas do módulo Compositor de Serviços: A Lista de Operações (1); Área de Composição (2); e Sugestão de Operação (3). No canto esquerdo encontra-se a lista de operações dos serviços web semânticos disponíveis para que o usuário inicie a criação de sua composição. A Região central é o espaço que o usuário tem para criar sua composição, sendo cada nó uma operação e a aresta entre nós as conexões entre entrada e saída das operações. Próximo à operação X encontra-se o resultado do algoritmo de sugestão de serviços para a composição até o momento, com foco na operação X.

Figura 15 – A interface gráfica do componente *Service Composer*.



A lista de operações do módulo Compositor de Serviços, como visto na seção anterior, é disponibilizada através do módulo Recuperador de Serviços. Para essa lista de operações, o módulo utiliza o valor do critério de disponibilidade para filtrar a lista de serviços apresentados. O intuito de aplicar essa métrica é apresentar uma lista de serviços com índice de disponibilidade igual ou superior ao valor mínimo aceitável. A razão para o refinamento das operações é apresentar serviços com um certo grau de confiabilidade. Pelo fato de considerar apenas o critério de disponibilidade, seu resultado não pode ser considerado como sugestão de serviços. Isso é feito pelo componente de sugestão de serviços que será apresentado na próxima seção.

3.5 O MECANISMO DE SUGESTÃO DE SERVIÇOS SEMÂNTICOS

O mecanismo de sugestão de serviços semânticos do S^3M encontra-se no componente *Matcher* de Serviços. Para seu funcionamento, o mesmo precisa da lista de operações disponibilizadas pelo Recuperador de Serviços e de outros parâmetros passados por outro componente da arquitetura, e apresentado nesta seção, o Analisador de Saídas.

O módulo Analisador de Saídas é o responsável por extrair os dados da composição sendo criada na componente Compositor de Serviços e passá-los como parâmetro para o *Matcher* de Serviços. Ele

extrai todas as saídas das operações contidas na composição, com exceção da operação selecionada (ou seja, que possui o foco do usuário), a qual é passada separadamente como parâmetro.

Parte do processo de sugestão consiste em verificar se as entradas e saídas de operações diferentes são compatíveis, o que é feito através do *match* semântico entre elas. Para isso utiliza-se a classificação proposta por Paolucci *et al.* (2002), representada pela função a seguir.

$$match(Output,Input)=\{x \mid x \in (exact, plugin, subsumption, fail)\}$$

Para cada grau de similaridade entre os conceitos ontológicos de entrada e saída, associa-se um valor que posteriormente será utilizado pelo algoritmo que calculará o grau de recomendação associado a uma operação. A tabela 2 apresenta os valores do grau de similaridade para cada tipo de *match*.

Tabela 2 – Valor associado a cada grau de similaridade.

Match	***Valor de x***
Exact	0
Plugin	1
Subsumption	2
Fail	3

O *match* entre operações é feito pelo componente *Matcher* de Serviços através de um algoritmo que compara uma lista de saídas com uma lista de entradas. O pseudocódigo do Algoritmo 2 apresenta como é feito o *match*. O uso de listas de entradas e saídas como parâmetros ao invés de operações se dá pelo fato de que a função pode ser utilizada tanto pela operação com foco, quanto com a lista de saídas da composição.

A função do algoritmo 2 percorre a lista de entradas verificando o *match* semântico com cada saída da lista pegando o menor valor, ou seja o mais próximo de um *match* exato. Se a entrada possuir um *match Exact* ou *Plugin* com alguma saída da lista, a entrada é adicionada à lista de *matches*. Por fim, a função retorna todas as entradas que satisfaçam esse critério.

Algoritmo 2 – *Match* Semântico entre entradas e saídas

```
01 function semanticMatch(outputList, inputList)
02   def matchList = ∅, mismatchValue = none
03   for each input in inputList do
04     mismatchValue = Match.FAIL
05     for each output in outputList do
06        mismatchValue = min(mismatchValue, SM(output, input))
07     end for
08     if mismatchValue in [Match.EXACT, Match.PLUGIN] then
09        matchList.add ( input )
10     end if
11   end for
12   return matchList
13 end
```

A principal função do componente *Service Matcher* é representada pelo pseudocódigo do Algoritmo 3. A função recebe 3 parâmetros. Conforme dito anteriormente, os parâmetros *focusOperation* e *OutputList* são parâmetros extraídos da área de composição de serviços pelo componente *Output Analyzer*. Na Figura 15, o parâmetro *focusOperation* é representado pela operação X, e a lista de saídas consiste das saídas da operação K e O.

Já a lista de operações – o parâmetro *operationList* do lgoritmo – é passada pelo componente *Service Retriever*, através da função apresentada no Algoritmo 1.

Algoritmo 3 – Sugestão de operações compatíveis com determinada operação e saídas adjacentes.

```
01 function semanticSuggestion(focusOperation, outputList,
operationList)
02   def suggestList = ∅
03   for each operation in operationList do
04      smFocus = semanticMatch(focusOperation.outputs,
                                  operation.inputs)
05      if smFocus.matchList ≠ ∅ then
06         smOutputs = semanticMatch(outputList,
                      operation.inputs – smFocus.matchList)
07         suitability = calcCSo(operation,
                      smFocus.matchList ∪ smOutputs.matchList)
08           suggestList.add(operation,  suitability)
09      end if
10  end for
11  return suggestList.sortBy(suitability)
12 end
```

A linha 3 do algoritmo mostra a lista de operações disponíveis sendo percorrida, verificando em seguida se a operação possui *match* semântico com a operação que possui o foco. Caso a lista não esteja vazia, linha 05, significa que ao menos uma entrada da operação sendo iterada possui um *match* satisfatório. Se isso acontecer significa que a operação analisada pode ser considerada como sugestão para o usuário.

O próximo passo é calcular o *match* semântico entre as entradas da operação que não tiveram um *match* satisfatório com a operação de entrada, e a lista de saídas da composição sendo criada. Isso ocorre na linha 06 do Algoritmo 3. Essa é a razão para utilizar a lista de entradas e saídas como parâmetro da função do Algoritmo 2, pois na segunda execução o match semântico não ocorre entra operações, mas entre uma lista de saídas e as entradas restantes de uma operação.

Na linha 07 ocorre o cálculo de adequabilidade da operação sendo iterada. Esse cálculo é representado pela equação a seguir.

$$GA_O = SF_O * \alpha + SR_O * \beta + (1 - d_O) * \gamma$$

onde:

- GA_O: Grau de Adequabilidade da operação *O*;
- SF_O: Similaridade Semântica entre as saídas da operação com foco e a entrada da operação O;
- SR_O: Similaridade Semântica entre as saídas das outras operações da composição e as entradas restantes da operação O;
- d_O: disponibilidade do serviço que fornece a operação *O*;
- α: peso associado ao *match* com a operação com Foco;
- β: peso associado ao *match* com as saídas da composição;
- γ: peso associado à disponibilidade do serviço.

Quanto menor o grau de adequabilidade de uma operação, mais compatível para a composição ela é considerada. O valor do grau de similaridade de entradas e saídas com *match Exact e Plugin* é de zero e um, respectivamente. Já o valor de disponibilidade de serviço com baixo período de *downtime* é mais próximo de 1, sendo assim é necessário subtrair de 1 o valor de disponibilidade para que o cálculo de adequabilidade retorne serviços com mais disponibilidade primeiro.

De volta ao algoritmo de sugestão de operações, na linha 8 o algoritmo adiciona à lista de sugestão a operação junto com o valor do grau de adequabilidade da operação. Por fim, na linha 11 o algoritmo retorna a lista de sugestões ordenada crescentemente.

3.6 PERSISTÊNCIA DAS COMPOSIÇÕES

Após a criação da composição, é possível que o usuário armazene a mesma. A persistência de tais informações permite a criação de uma base de dados com todas as composições já criada pelos usuários do S^3M.

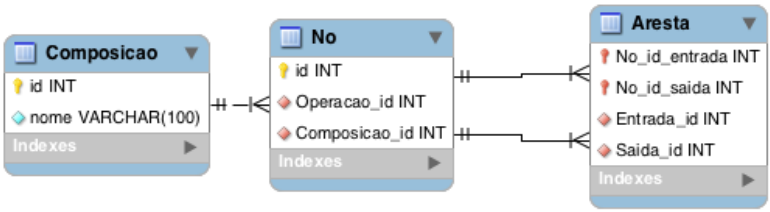
Isso permitirá que no futuro o mecanismo de sugestão leve em consideração as decisões anteriores dos usuários. Fazendo com que a sugestão passem a ser baseadas na similaridade semântica e centrada nas escolhas dos usuários.

Os componentes da arquitetura que fazem parte do processo de persistência de composições são: o registo de composições; e o repositório de composições.

O módulo registro de composições é responsável por extrair da área de criação as informações da composição e persisti-las no repositório.

O repositório, conforme mostra a Figura 16, é composto por um conjunto de tabelas que fazem parte do mesmo *schema* de banco de dados da Figura 14. As tabelas foram modeladas para remeter o conceito de que as composições criadas são grafos. Omitiu-se da figura as informações sobre as tabelas Operação, Entrada, e Saída, pois as mesmas foram apresentadas na Figura 14.

Figura 16 – Modelo entidade-relacionamento do repositório de composição



Uma composição possui um nome próprio, seguido por um ou mais nós associados a ela. Todo nó possui o mapeamento da operação que o mesmo representa. Já a relação entre nós é mapeada pela tabela Aresta, que armazena as informações dos nós que se relacionam e qual a saída e entrada que se conectam nessa relação.

4 IMPLEMENTAÇÃO E AVALIAÇÃO DA ABORDAGEM

O capítulo anterior apresentou os principais componentes e funcionalidades da abordagem proposta. Neste capítulo serão apresentados detalhes da implementação do S^3M, seguidos de testes realizados com a implementação do mecanismo, com o objetivo de verificar e avaliar a abordagem desenvolvida.

Os detalhes da implementação do S^3M são apresentados na seção 4.1. A seção 4.2 apresenta o CVFlow, ferramenta de composição de fluxos escolhida para integrar a arquitetura proposta. Os testes de performance e simulação de cenários específicos que demonstram o comportamento da proposta serão expostos na seção 4.3. Por fim, a seção 4.4 compara a proposta deste trabalho com trabalhos correlatos existentes.

4.1 DETALHES DE IMPLEMENTAÇÃO

A composição automatizada de serviços não extingue a interação com o usuário, que ainda se faz necessária para especificar as entradas e saídas esperadas da composição que a ferramenta automatizará. Sendo assim, optou-se em não utilizar como ponto de partida uma proposta totalmente automatizada de composição, como é o caso do trabalho de Hobold e Siqueira (2012), mas sim o de automatizar o processo de composição de uma ferramenta manual através do uso do mecanismo de sugestão proposto.

Através dos trabalhos correlatos, buscaram-se características comuns das abordagens de composições de serviços e também possíveis abordagens diferentes para os trabalhos existentes. Assim, chegou-se à arquitetura apresentada no capítulo anterior, que emprega uma abordagem para composição semiautomática de serviços através de sugestão de operações semanticamente compatíveis e que levam em conta a disponibilidade dos serviços sendo compostos.

Em seguida, procurou-se implementar os componentes propostos e, principalmente, criar um protótipo para demonstrar o funcionamento deste mecanismo de sugestão de serviço. Tendo em vista que o mecanismo de sugestão tem como objetivo aumentar a automação do processo de sugestão, buscou-se uma aplicação de composição manual de serviços. Para isso, a ferramenta escolhida para integrar a abordagem proposta foi o CVFlow – ferramenta desenvolvida pelo grupo de pesquisa Lapix da Universidade Federal de Santa Catarina, que será detalhado na próxima seção. Apesar de não se tratar de uma ferramenta

de composição de serviço e, sim de uma ferramenta de composição de fluxo, o conceito por trás de ambos é similar, visto que ambas as formas são análogas à composição de grafos.

Paralelamente à criação do protótipo, integrando a ferramenta de composição com a abordagem proposta, realizaram-se testes de performance para avaliar a afirmação de Laga, Bertin e Crespi (2009) de que a diminuição do grau de automação do processo de composição reduziria o gargalo de processamento.

Para a implementação dos componentes do S[3]M utilizou-se a linguagem de programação Java, e o framework de persistência de dados Hibernate[3] para o mapeamento das entidades dos repositórios SAWSDL e de Composição. A escolha da linguagem tem influência da ferramenta de composição de workflow CVFlow, que também é desenvolvida utilizando a linguagem Java.

Para a extração das informações do serviços SAWSDL, estudou-se a utilização de framework usados em outras abordagens, como é o caso do framework EasyWSDL[4], utilizado no trabalho de Hobold e Siqueira (2012). Porém, a falta de uma biblioteca bem definida, somada à dificuldade de fazer com que os testes unitários fossem satisfatórios, fez com que fosse abandonado o uso de tal framework no protótipo criado. Ao invés disso, utilizou-se a biblioteca de manipulação de arquivos XML do Java, o DOM[5], já que os arquivos WSDL utilizam o formato XML para descrever seus serviços.

Para armazenar os dados dos repositórios de SAWSDL e de composições, usou-se a ferramenta de gerenciamento de banco de dados MySQL[6]. Conforme dito anteriormente, foi adotado o framework Hibernate para o manuseio dos objetos persistidos.

Para lidar com as ontologias do serviços semanticamente anotados e principalmente realizar o *match* entre operações foram utilizados os frameworks OWL API[7] e HermiT[8]. O framework OWL-API possui recursos que possibilitam a criação, manipulação e serialização de ontologias OWL. Suas versões mais recentes foram desenvolvidas para suportar apenas ontologias em OWL 2. Já o framework HermiT é uma ferramenta de raciocínio de ontologia OWL. Com ele é possível verificar se uma ontologia é ou não consistente, e

[3] http://hibernate.org/
[4] http://easywsdl.ow2.org/extensions-sawsdl.html
[5] https://docs.oracle.com/javase/tutorial/jaxp/dom/
[6] https://www.mysql.com/
[7] http://owlapi.sourceforge.net/
[8] http://hermit-reasoner.com/

identificar o relacionamento entre classes de ontologias. Os dois frameworks são usados pelo componente de *Matching* de serviços da arquitetura proposta.

Desta forma, empregando este conjunto de linguagens e frameworks, foi realizada a implementação dos componentes da arquitetura proposta no capítulo anterior, resultando na implementação do protótipo do S^3M. Na seção 4.2 é apresentada a ferramenta CVFlow, seguido das mudanças realizadas na mesma para que fosse possível integrar o mecanismo de sugestão deste trabalho, expondo o resultado final de tal integração.

4.2 CVFLOW

Conforme destacado anteriormente, um dos objetivos deste trabalho é aumentar a automação de uma ferramenta manual de composição. Nesse caso optou-se por uma ferramenta desenvolvida por um grupo de pesquisa da própria Universidade Federal de Santa Catarina. O CVFlow viabiliza recursos para criação de *workflows* através da construção de grafos. De acordo com Prazeres (2009), a composição de serviços ou de *workflows* é análoga ao comportamento de grafos. Os serviços correspondem aos nós de um grafo. Enquanto as arestas representam a relação entre a entrada e saída de dois serviços. No caso dessa abordagem, essas ligações levam em conta a similaridade semântica entre entradas e saídas.

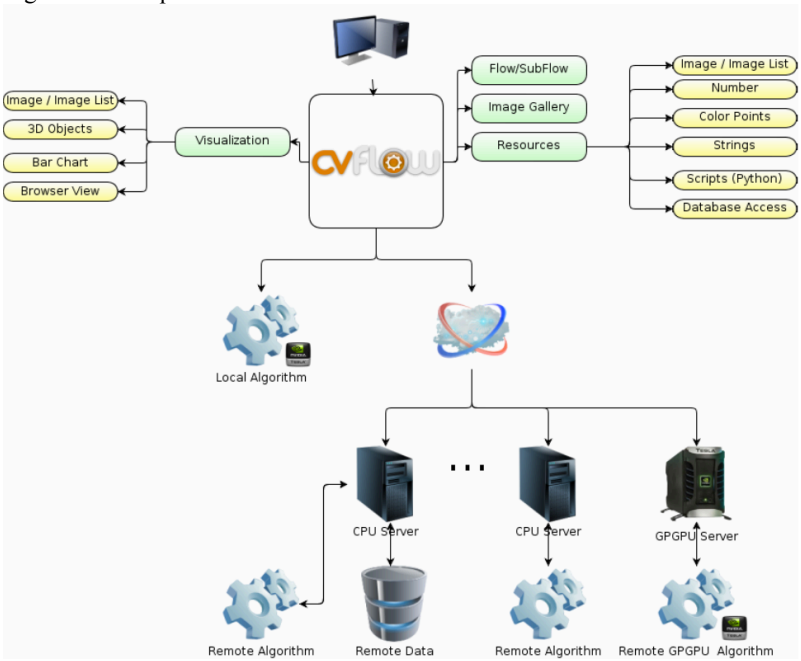
Originalmente, o CVFlow foi desenvolvido pelo Laboratório de Processamento de Imagem e Computação Gráfica (Lapix), com o intuito de auxiliar a criação de fluxos de processamento de imagem. Apesar de ter sido criada com esse objetivo, a ferramenta foi projetada para permitir múltiplos usos e ser estendida para atender outros propósitos. Alguns dos exemplos de extensões da ferramenta que o laboratório vêm desenvolvendo são controle de hardware para a construção de imagens 3D e processamento de vídeo (WEBER, 2013).

A Figura 17 apresenta a arquitetura do CVFlow segundo Weber (2013). O usuário interage diretamente com a ferramenta criando *workflows*, utilizando recursos, ferramentas de visualização, subfluxos e algoritmos.

- Os recursos (*Resource)* são nós que representam tipos de entradas para os algoritmos do CVFlow. Nesse caso podem ser: imagens; números; pontos de cor; strings; até mesmo *Scripts* ou valores extraídos de banco de dados.

- Os componentes de visualização (*Visualization*) auxiliam o usuário a ver os resultados dos algoritmos de manipulação de imagem executados na composição.
- No CVFlow um fluxo pode ser salvo e posteriormente ser utilizado como um subfluxo para um novo fluxo sendo criado. Esse subfluxo é visto como um simples nó para o novo fluxo. O usuário precisa inserir todas as entradas necessárias e o subfluxo irá retornar todas as saídas que ele produziu.
- Os algoritmos do CVFlow podem ser locais e remotos. Os algoritmos locais são executados na mesma máquina onde o CVFlow está instalado. Já algoritmos remotos são executados em servidores que possuem instâncias do CVFlow instaladas, por exemplo: servidores do Lapix ou Sharcnet[9].

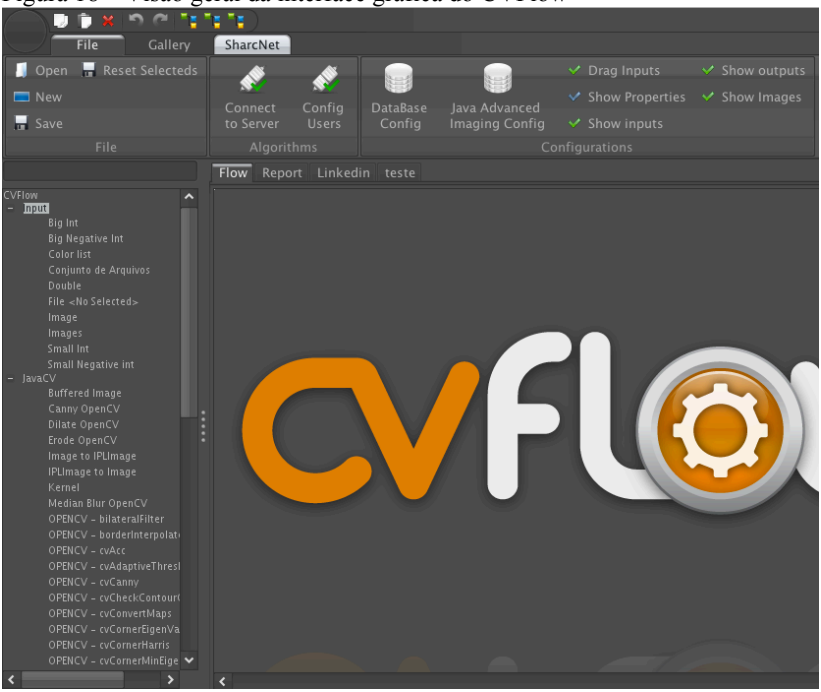
Figura 17 – Arquitetura do CVFlow



[9] https://www.sharcnet.ca/my/about

A interface gráfica do CVFlow é similar à de outras ferramentas de composição de fluxo como a apresentada por Monteiro (2008) e Pi et al. (2012), e também similar à interface do componente de composição da proposta do S^{3}M, representado na Figura 15. A Figura 18 apresenta a interface gráfica do CVFlow.

Figura 18 – Visão geral da interface gráfica do CVFlow

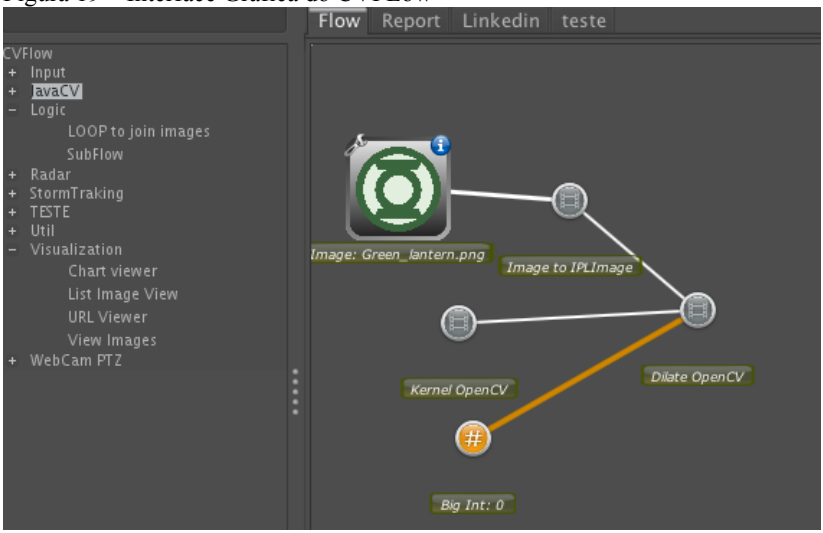


A parte superior do CVFlow é composta por um painel de controle onde é possível salvar, criar uma nova, ou carregar composições, uma área responsável pela configuração dos algoritmos que são executados remotamente, configurações de banco de dados e de visualização da composição, e também, uma aba separada para configuração de algoritmos remotos na Sharcnet. A Figura 18 mostra o painel de configuração do CVFlow.

A Figura 19 mostra um exemplo de composição de *workflow* no CVFlow. No canto esquerdo da figura encontra-se a lista de recursos e algoritmos que o CVFlow disponibiliza pra criação de seus fluxos. O exemplo de composição da Figura 19 começa com um nó de visualização de imagem, que serve como parâmetro para o algoritmo de

transformação de imagens para o formato padrão utilizado pelos algoritmos de processamento do CVFlow. Por fim, é realizada uma operação de Dilatação da Imagem, que recebe outros dois parâmetros para sua correta execução. Ao clicar no algoritmo de Dilatação todo o fluxo é executado. A funcionalidade de execução do CVFlow não fará parte da Integração com o S^{3}M, tendo em vista que não é o objetivo desta proposta a execução das composições de serviços criadas.

Figura 19 – Interface Gráfica do CVFLow



O processo de modificação da ferramenta para suportar o mecanismo de sugestão semântica deste trabalho é visto na subseção 4.2.1.

4.2.1 Integração entre CVFlow e S^{3}M

A interação com o usuário compositor da arquitetura do S^{3}M acontece no componente de composição de serviços da proposta. Porém, ao invés de implementar a interface gráfica do componente, utilizou-se o CVFlow para isso. Esta seção apresenta detalhes da integração entre ferramentas.

Apesar de não se tratar de uma ferramenta de composição de serviços, o CVFlow ainda pode ser classificado como uma ferramenta manual de *workflows* dinâmicos. Uma ferramenta manual pois não

possui nenhum mecanismo de automação do processo de composição dos serviços. E uma ferramenta dinâmica já que os fluxos podem ser executados conforme são criados, podendo sofrer modificações posteriores.

Com a integração da ferramenta ao S^3M, ela passa a ser uma ferramenta semiautomática de composição de serviços. Com isso, o mecanismo de sugestão de serviços auxilia o usuário no processo de criação das composições, empregando uma abordagem dinâmica, já que as sugestões podem ser diferentes conforme o momento em que a composição está sendo criada, considerando-se que a variável de disponibilidade poderá ser diferente.

Realizou-se a integração em um Macbook Pro com CPU Intel Core 2 Duo 2.4GHz, 8 GB de RAM, com o sistema operacional Windows 7. A razão para não utilizar o sistema operacional da Apple é porque o CVFlow utiliza bibliotecas Java que possuem dependências com bibliotecas nativas que não funcionaram ao configurar o ambiente no OS X.

De posse do código-fonte do CVFlow, deu-se início ao processo de adaptação e integração com o S^3M. De princípio eliminou-se da ferramenta sua capacidade de execução de algoritmos, pois a proposta deste trabalho visa à criação de composição e não a execução das mesmas.

O segundo passo foi estender a classe que representa os algoritmos do CVFlow, para que esse armazene internamente uma referência para a operação que o mesmo representará. O nome do algoritmo passará a ser o {nome do serviço}_{nome da operação}.

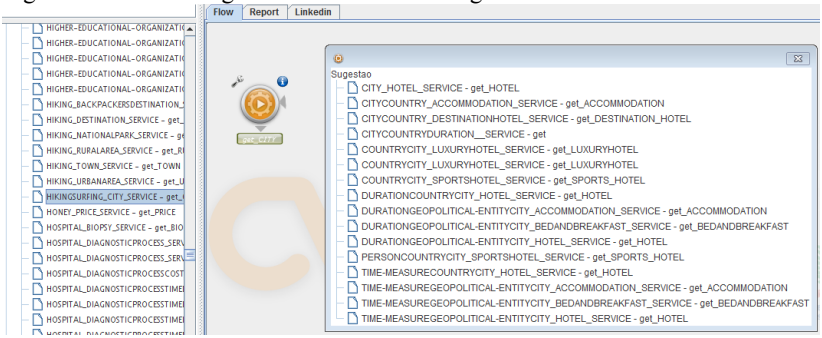
Outra mudança no nó Algoritmo do CVFlow é que o mesmo deixará de receber e retornar recursos como parâmetros de entrada e saída, e passará a refletir a entrada e saída da operação que está internamente referenciada. Ou seja, eliminam-se os diversos tipos de recursos existentes, e passa a existir apenas um nome e a URI para a ontologia da entrada ou saída da operação.

A Lista de operações passadas pelo componente de Recuperador de Serviços do S^3M precisa passar por um processo de adaptação para torna-se uma lista de algoritmo do CVFlow. Assim é possível adicionar a lista de operações à interface gráfica no canto esquerdo da mesma.

O único componente que não será uma adaptação das funcionalidades existentes do CVFlow é a janela de sugestão de serviços. O resultado final da adaptação encontra-se nas figuras a seguir. A Figura 20 apresenta o resultado da integração do S^3M com a interface do CVFlow. Nela é possível notar a lista de operações, que são as

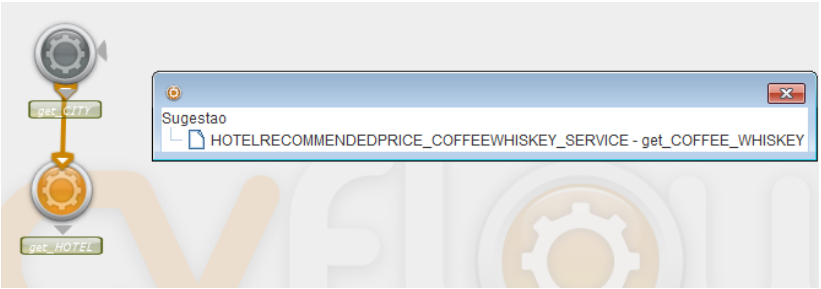
mesmas utilizadas no experimento de performance da seção 4.3.1, e também a janela de sugestão para a primeira operação da composição.

Figura 20 – Interface gráfica do CVFLow integrado ao S^{3}M



Já na Figura 21 é possível visualizar com mais detalhes a área de composição, junto com a sugestão de operação para o segundo nó da composição.

Figura 21 – Interface gráfica da área de composição do CVFLow integrado ao S^{3}M



Outra decisão de projeto que ficou evidente nas Figuras 20 e 21, foi optar por um esquema de cores mais claras e similar à utilizada por programas do Windows na interface gráfica do CVFlow.

A ferramenta serviu como maneira de interagir graficamente com o usuário, cobrindo o papel do componente Compositor de Serviço apresentado na Figura 13. A não utilização da mesma impactaria apenas na necessidade de implementação de uma nova interface com o usuário compositor, os outros componentes da arquitetura do S^{3}M permaneceriam inalterados.

4.3 AVALIAÇÃO DA PROPOSTA

Com o intuito de avaliar o protótipo desenvolvido para o S^3M foram realizados dois experimentos distintos com o protótipo. O primeiro experimento visa verificar o desempenho das composições em cenários extremos. Já a segunda avaliação consiste de analisar o comportamento do mecanismo de sugestões num cenário controlado de operações, verificando as operações retornadas em cada cenário e discutindo se a ordem dos resultados é satisfatória.

4.3.1 Testes de Desempenho

Os experimentos foram realizados em um Macbook Pro com CPU Intel Core 2 Duo 2.4GHz, 8 GB de RAM, com o sistema operacional OS X 10.9 Maverick, utilizando a versão do Java JDK 1.8 e o banco de dados MySQL na versão 5.6.

A base de dados dos serviços foi a mesma utilizada por Hobold e Siqueira (2012), apenas adicionando o valor 1.0 para a coluna de disponibilidade de tais serviços. Os serviços usados foram obtidos através do repositório www.semwebcentral.org. Para armazenar as ontologias desses serviços utilizou-se o servidor Apache Tomcat 8.0.

O repositório é constituído de 1000 exemplos de serviços semanticamente anotados. Para os testes realizados replicou-se o número de serviços para analisar o comportamento da ferramenta com uma maior carga.

Os testes realizados consistiram da execução sequencial da mesma quantidade de operações sobre o mecanismo de sugestão de serviços invocando uma mesma operação para todos os casos. Registrou-se o tempo que o mecanismo leva para fazer o *match* semântico entre a operação com foco e a operação candidata a ser sugerida, o *match* com as outras saídas, o tempo que leva para realizar o cálculo de similaridade semântica, o processamento e o tempo total de execução. Processamento é considerado toda a operação que o algoritmo realiza que não seja as três primeiras operações citadas. Para chegar a esse resultado subtrai-se do tempo total o tempo gasto para fazer o *match* semântico entre a operações com foco e as outras saídas, e o cálculo de similaridade semântica.

Cada teste consistiu de 1250 execuções, dividas em cinco intervalos de 250 execuções sequencias da operação para identificar a média de valores apresentado na Tabela 3 e também poder calcular o

desvio padrão e a mediana em cada um dos testes. Os valores apresentados na Tabela 3 encontram-se em milissegundos (*ms*).

Tabela 3 – Tempo gasto no processo de sugestão do S^3M

	1000 serviços (ms)	**2000 serviços (ms)**	**3000 serviços (ms)**	**4000 serviços (ms)**	**5000 serviços (ms)**
Semantic Mismatch Operação com foco	92.08	161.10	217.63	281.95	349.67
Semantic Mismatch Outras Saídas	0.01	0.02	0.03	0.04	0.05
Calc. Similaridade Semântica	0.02	0.03	0.04	0.04	0.05
Processamento	42.03	33.19	31.75	32.68	33.01
Total de Tempo	134.12	194.29	249.39	314.64	382.78
Desvio Padrão	50.79	55.35	57.17	62.63	52.42
Mediana	122.55	184.43	239.63	304.75	374.82

A razão para que o teste de performance não utilize outras operações no processo de composições é porque o repositório de operações utilizados é composto quase exclusivamente por operações que possuem apenas uma entrada e uma saída. Por isso, os baixos valores no tempo para o *match* semântico com as saídas das outras operações da composição.

Outro processo com baixo custo computacional é o de cálculo da similaridade semântica, já que este consiste em operações aritméticas simples.

Grande parte do tempo desprendido para a realização do processo de sugestão de serviço vem do *match* semântico da operação com foco. No caso dos testes envolvendo 1000 serviço, um terço do tempo total advém do processamento de outras operações. Pode-se observar que conforme a carga de serviços aumenta, os valores de processamento permanecem estáveis.

A Figura 22 apresenta o gráfico de distribuição com o total de tempo de execução dos testes realizados, somado ao desvio padrão de cada um dos cenários de teste. O gráfico demonstra que o tempo de execução tem um crescimento linear (O(*n*)) conforme o número de serviços aumenta.

A Figura 23 mostra a distribuição dos tempos de cada um dos experimentos realizados. Nela é possível notar que o tempo de execução

dos testes com serviços iguais apresentam, em sua maioria, um tempo similar, com apenas alguns casos de testes que se distanciam um pouco mais dos valores médios obtidos.

Figura 22 – Tempo de execução do mecanismo de sugestão nos experimentos.

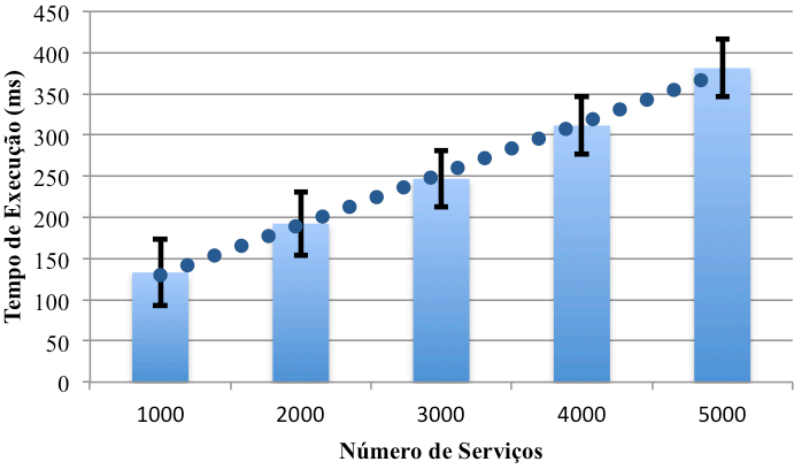
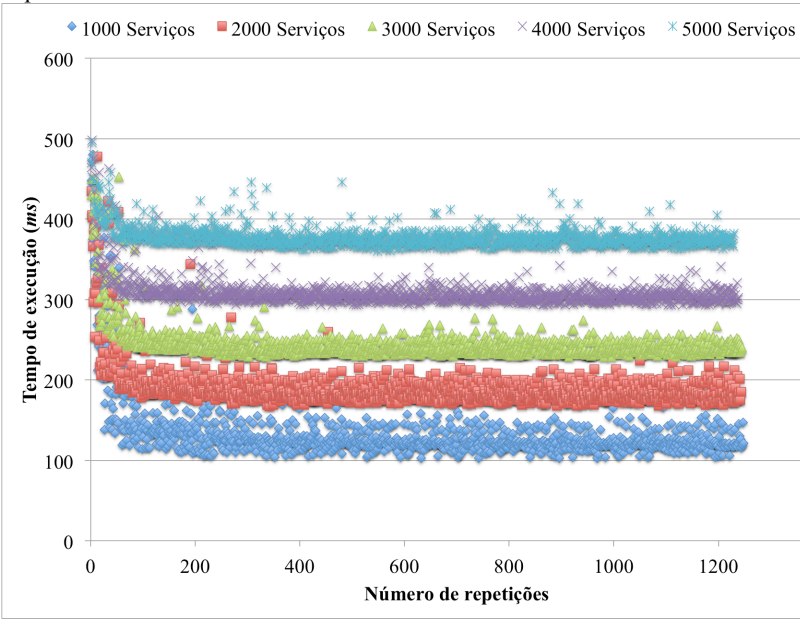


Figura 23 – Distribuição do tempo de execução do mecanismo de sugestão nos experimentos.

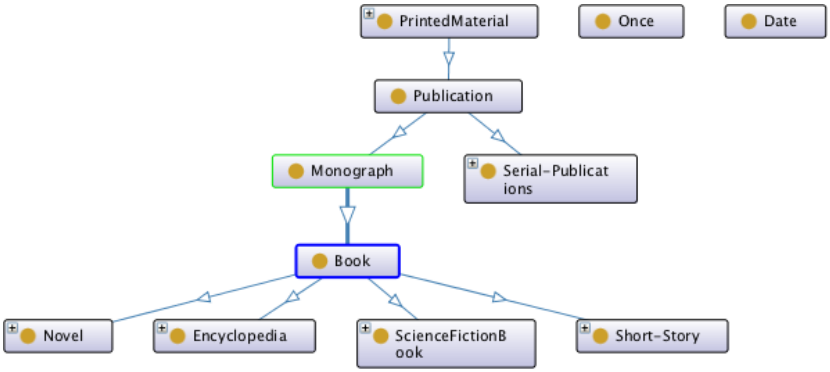


4.3.2 Simulação

Os experimentos realizados nesta seção visam analisar o comportamento da proposta em cenários controlados. Para isso utiliza-se a ontologia *books.owl*, retirada dos serviços do repositório da www.semwebcentral.org. A referida ontologia encontra-se disponível na integra no Anexo A que encontra-se no fim deste trabalho.

A Figura 24 é uma representação gráfica de uma parcela da ontologia utilizada nas simulações. As subclasses de *Book* e sua superclasse *Monograph* serão utilizadas nas operações de cada cenário simulado. No caso do cenário 2 as outras classes ontológicas utilizadas ou possuirão um *match exact* ou não haverá *match* com as saídas da composição. Dessa forma, foram omitidas da Figura 24.

Figura 24 – Grafo contendo parte da ontologia *books.owl*



Criaram-se cenários específicos com um número limitado de operações, tanto na composição, quanto disponíveis para serem sugeridas. Em cada cenário é descrita cada operação existente no repositório, seguida do resultado que o algoritmo de sugestão retornará com uma breve análise dos resultados obtidos.

Para realizar a simulação, adotaram-se os valores contidos na Tabela 4 para as constantes da função que realiza o Cálculo da Similaridade Semântica. Os valores foram escolhidos a fim de priorizar um melhor *match* semântico entre as operações que possuam um maior *match* com a operação com o foco, seguida das outras saídas da composição e, por fim, a disponibilidade de seus serviços.

Tabela 4 – Valores das constantes da fórmula de similaridade

Constante	Valor
α	3
β	2
γ	1

O valor da variável global que determina o mínimo de disponibilidade de serviços aceitável é de 0.5 para todos os cenários simulados.

4.3.2.1 Cenário 1

O primeiro cenário consiste de apenas uma operação na composição sendo criada. Já no repositório de operações encontram-se 3 operações que possuem como entrada um match *exact* com a saída da operação, porém com valores de disponibilidades diferentes. Há ainda uma operação com valor de disponibilidade 1, porém sua entrada é um *plugin* da saída da operação com foco. A Tabela 5 apresenta em sua primeira linha a operação com foco na composição sendo criada, seguida nas próximas linhas pelas operações existentes no repositório de serviço. Cada operação possui seus valores de entradas, suas entradas e saídas e o valor de disponibilidade de serviços que as contém.

Tabela 5 – Operações do Cenário 1

Operação	Entrada	Saída	Disponibilidade
highPriceBook	-	Book	1
getAuthor(1)	Book	Author	1
getAuthor(2)	Book	Author	0.75
getAuthor(3)	Book	Author	0.40
getAuthor(4)	Monograph	Author	1

A execução do mecanismo de sugestão da proposta retorna as operações descritas na Tabela 6, junto com o valor de similaridade calculado. O mecanismo opta por retornar primeiramente as operações que possuem um *match exact* entre as entradas e saídas. A operação *getAuthor(2)* é retornada primeiramente que a operação *getAuthor(4),* mesmo possuindo uma disponibilidade inferior a esta, devido ao fato de possuir um *match plugin*. A operação *getAuthor(3)* é eliminada do

processo de sugestão por possuir um taxa de disponibilidade inferior ao mínimo aceitável.

Tabela 6 – Resultado da sugestão de serviço do cenário 1

Operação	Cálculo Similaridade
getAuthor(1)	0
getAuthor(2)	0.25
getAuthor(4)	3

4.3.2.2 Cenário 2

O objetivo do experimento do segundo cenário é verificar o comportamento do mecanismo no caso de existirem dois valores de entrada nas operações sendo avaliadas como possíveis sugestões. Nesse teste o critério de disponibilidade dos serviços serão desconsiderados, ou seja, todos os serviços terão valor 1 de disponibilidade.

Além disso, a composição simulada possui outras operações, sendo que uma delas possui como saída a classe *Novel*. A Tabela 7 apresenta as operações utilizadas nessa simulação, sendo que a primeira linha é a operação com foco da simulação.

Tabela 7 – Operações do Cenário 2

Operação	Entrada	Saída	Disponibilidade
getBook	Title	Author | Book	1
getPrice(1)	Author | Monograph	Price	1
getPrice(2)	Author | Novel	Price	1
getBook	Author | Grade	Book	1

A primeira operação *getPrice* possui um *match exact* para o parâmetro *Author* e um *match plugin* entre *Monograph* e *Book*. Dessa forma não é necessário verificar o *match* da operação com as outras saídas da composição.

Já a segunda operação possui apenas um *match* aceitável com a operação de foco. Assim ela continua sendo utilizada como operação sugestível, pois o algoritmo de sugestão necessita de no mínimo um *match* com a saída da operação com foco, o segundo valor de entrada possui um *match exact* com uma das saídas da composição.

Por último, a terceira operação possui um *match exact* com a operação com foco, porém não há saída na composição para o segundo parâmetro de entrada. Mesmo assim, a operação é considerada

sugestível, já que há ao menos uma entrada satisfeita com uma saída da operação com foco.

A Tabela 8 apresenta a ordem em que as operações seriam retornadas para o usuário. A operação *getPrice(2)* é a primeira da lista por ser a única operação a possuir *match exact* para suas duas entradas, mesmo não sendo todas provenientes da operação com foco. Já no segundo caso, de maneira similar ao primeiro cenário, possui uma das entradas com *match plugin* com a operação com foco. Por fim, na lista de sugestão está a operação *getBook* que não possui um *match* satisfatório para uma de suas operações, por isso retorna o valor 6. O *match exact* de *Author* gera o valor 0 para a primeira parte da fórmula. Como o segundo parâmetro não é satisfeito nem com as saídas da composição, o mesmo recebe o valor associado a um *fail(3),* e como o valor de β é 2, o cálculo retorna o valor da tabela 8.

Tabela 8 – Resultado da sugestão de serviço do cenário 2

Operação	**Cálculo Similaridade**
getPrice(2)	0
getPrice(1)	3
getBook	6

4.4 COMPARAÇÃO COM TRABALHOS RELACIONADOS

Após apresentar a arquitetura proposta no capítulo anterior e apresentar neste capítulo detalhes da implementação deste trabalho, é possível fazer uma comparação com os trabalhos correlatos descritos anteriormente na seção 2.5. Os aspectos em que este trabalho difere dos outros serão apresentados a seguir.

O trabalho de Pi et al. (2012) é o que mais se assemelha com a abordagem proposta. Além de realizar as composições de maneira semiautomática, o trabalho também possui um módulo para transformar *Web Services* em serviços web semânticos. As diferenças estão no tipo de composição e também na linguagem escolhida para representar os serviços web semânticos, Pi et al. (2012) opta pelo OWL-S como linguagem, enquanto o trabalho apresentado suporta a linguagem que é padrão W3C para serviços web semânticos, o SAWSDL.

O trabalho de Meiyu e Maoji (2013) não realiza implementação, apenas analisa métodos de composição de serviços web semânticos. O framework proposto é um esboço e não referencia qualquer linguagem, tecnologia ou métodos. Apesar disso, o trabalho teve grande influência

sobre a arquitetura proposta neste trabalho, sendo assim recebe destaque entre outros trabalhos correlatos.

O trabalho de Monteiro (2008) assemelha-se à proposta deste trabalho. Os serviços são compostos de forma semiautomática, através de uma interface gráfica em que o usuário é guiado através da montagem da composição de serviços, por meio de sugestões de serviços que são passíveis de integração. A composição desse trabalho não considera critérios de *QoS*, porém este possui loops e operações condicionais, características não apresentadas no nosso trabalho.

O trabalho de Hobold e Siqueira (2012) adota uma abordagem automática para composição de serviços, e utiliza a mesma linguagem de representação de serviços web semânticos, o SAWSDL. Aspectos referentes à persistência das descrições dos serviços serviram de base para a proposta deste trabalho.

Dos trabalhos apresentados, a proposta de Hobold (2012) é o único que possui informação quanto ao tempo de execução em teste de performance. E o fato de que a proposta desse trabalho teve acesso à base utilizada em seus testes permitiu verificar se a afirmação de Laga, Bertin e Crespi (2009), de que um maior envolvimento com usuário diminuiria o tempo de construção da composição.

O que acontece nos testes feitos por Hobold é perceber que o gargalo da aplicação encontrava-se na construção de um grafo que satisfizesse os critérios estipulados. No caso do repositório de 1000 serviços, para chegar a composição desejada, levava-se aproximadamente 13 segundos para obter a melhor resposta, uma composição contendo 4 operações.

No trabalho apresentado, se levarmos em conta o tempo que o mecanismo leva para sugerir uma operação em um repositório de 1000 serviços, e multiplicando esse tempo médio por quatro obtemos o valor de 536.48 milissegundos para obter quatro sugestões.

Esse valor é obtido sem considerar o tempo que leva para o usuário adicionar graficamente (“arrastar”) os serviços na composição, e os testes foram feitos em máquinas diferentes. Mesmo assim, é possível afirmar que uma abordagem semiautomática de composição possui um custo computacional menor ao de uma abordagem totalmente automatizada de composição de serviços.

A Tabela 9 sumariza a comparação entres os trabalhos correlatos e a abordagem apresentada neste trabalho.

Tabela 9 – Comparação com a Proposta Apresentada

	Pi et al. (2012)	Meiyu e Maoji (2013)	Monteiro (2008)	Hobold e Siqueira (2012)	Silva e Siqueira (2014).
Linguagem	OWL-S	-	OWL-S	SAWSDL	SAWSDL
Forma de Composição	Semiautomática	-	Semiautomática	Automática	Semiautomática
Momento de Composição	Requisição	-	Desenvolvimento	Requisição	Requisição
Tipo de Composição	Estática	-	Estática	Dinâmica	Dinâmica
Técnica de Composição	Interação com o usuário	-	Interação com o usuário	Grafos	Interação com o usuário
Parâmetros de QoS	Não	-	Não	Não	Sim

5 CONCLUSÃO E TRABALHO FUTUROS

As abordagens para composições de serviços podem variar quanto ao grau de automação escolhido. Ferramentas totalmente automatizadas têm a vantagem de menor interação com usuário, maior custo computacional e detalhamento da ontologia utilizada. Já abordagens manuais possuem um baixo custo computacional, porém um envolvimento maior com o usuário, fazendo com que este tenha conhecimento maior sobre os serviços que fazem parte da composição.

Na abordagem deste trabalhou optou-se pela abordagem semiautomática para a criação de composição. A adoção dessa abordagem visa simplificar o trabalho do usuário sem ocasionar o alto custo computacional observado na abordagem automatizada.

A criação da composição é realizada empregando um mecanismo de sugestões de serviços que calcula a similaridade semântica das operações envolvidas e a disponibilidade do serviço da operação candidata a sugestão. Isso permite a sugestão dos serviços que possuem uma maior coesão semântica com a composição sendo criada e maior disponibilidade para ser invocada.

Vale ressaltar que o *match* semântico é feito entre entradas e saídas das operações. Visando buscar interconexão entre operações, a decisão de integrar a operação à composição ainda permanece nas mãos do usuário, que analisará qual das operações sugeridas melhor adequa-se à composição sendo criada.

As sugestões feitas pelo mecanismo serão dependentes diretamente das ontologias associadas ao serviços cadastrados no repositório SAWSDL. Por essa razão é difícil analisar e avaliar qualitativamente a proposta deste trabalho, tendo em vista que a qualidade das sugestões dependerá da descrição semântica dos serviços que o usuário fornecedor irá cadastrar.

Os testes de desempenho realizados com o protótipo criado evidenciam uma diminuição do tempo de execução. A razão encontrada para essa diminuição está no fato de a abordagem deste trabalho buscar sugestões de operações para complementar a composição, enquanto abordagens completamente automatizadas precisam criar grafos e realizar buscas em profundidade pra satisfazer os objetivos especificados.

5.1 CONTRIBUIÇÕES

Com o trabalho aqui apresentado, junto com os objetivos traçados é possível identificar as seguintes contribuições:

- Um mecanismo de sugestão de serviços baseados na similaridade semântica entre entradas e saídas da operações e critérios de disponibilidade dos mesmos;
- Algoritmo de cálculo de disponibilidade baseado em média ponderada entre a verificação mais recente e a média de verificações anteriores;
- Um protótipo que implementa a proposta de mecanismo de sugestão de serviços;
- Um protótipo que integra o mecanismo de sugestão de serviços proposto com uma ferramenta de composição de serviços, resultando em uma ferramenta gráfica que permite ao usuário criar composições auxiliado pelas sugestões do mecanismo.

5.2 TRABALHOS FUTUROS

A partir do que foi desenvolvido neste trabalho, identificam-se algumas possibilidades para dar continuidade ao que foi desenvolvido:

- Realizar testes com os usuários, a fim de popular o repositório de composições para posterior inclusão ao cálculo de sugestão a porcentagem de ocorrência em que uma operação foi inclusa a outra;
- Realizar estudos com usuários, verificando se os mesmos consideram benéfico o uso do mecanismo de sugestão semântica;
- Realizar a implementação do componente de composição de serviços em uma interface web para que as composições possam ser executadas remotamente em cima de uma instância única da implementação;
- Estender a arquitetura do S^3M, para que seja capaz de executar as composições criadas pelo mesmo.

5.3 PUBLICAÇÃO

A partir da abordagem proposta neste trabalho publicou-se o seguinte artigo:

- **Título:** The Use of Semantic-based Suggestions for Web Service Composition;
- **Congresso:** The 2014 International Conference on Semantic Web and Web Services (SWWS);
- **Local:** Las Vegas, Nevada, EUA;
- **Data:** 21-24 de julho de 2014;
- **Autores:** Diogo Phelipe Busanello da Silva, Frank Siqueira;
- **Estrato Qualis/CAPES:** A2.

Além do *short paper*:

- **Título:** S^3M: A Suggestion Mechanism for Semantic Web Service Composition;
- **Congresso:** Simpósio Brasileiro de Sistemas Multimídia e Web (WebMedia);
- **Local:** Manaus, Amazonas, Brasil;
- **Data:** 27-30 de outubro de 2015;
- **Autores:** Diogo Phelipe Busanello da Silva, Mathias Henrique Weber, Frank Siqueira;
- **Estrato Qualis/CAPES:** B3.

REFERÊNCIAS

BERNERS-LEE, Tim; HENDLER, James; LASSILA, Ora. The Semantic Web. **Scientific American,** [SI], p 28-37, 2001.

BENNER-LEE, Tim. **WWW2005 Keynote (2005)**. Disponível em: <http://www.w3.org/2005/Talks/ 0511-keynote-tbl/>. Acesso 25 de julho de 2015.

BIKAKIS, Nikos, et al. The XML and Semantic Web Worlds: Technologies, Interoperability and Integration. A Survey of the State of the Art. In: ANAGNOSTOPOULOS, Ioannis E., et al. **Semantic Hyper/Multimedia Adaptation**. Berlin: Springer, 2013. P. 319-360.

BOOTH, David, et al. **Web Services Architecture**. [SI], 2004. Disponível em: <htttp://www.w3.org/TR/ws-arch>. Acesso 7 setembro 2013.

BOUZID, Sara; CAUVET, Corine; PINATON, Jacques. **A survey of semantic web standards to representing knowledge in problem solving situations**. International Conference On Information Retrieval & Knowledge Management, vol., no., p.121-125, 2012

CARDOSO, Jorge. **Semantic Web Services: Theory, Tools and Applications**. Herhey, PA: IGI Global, 2007.

DUSTDAR, Schahram; SCHREINER, Wolfgang. **A survey on web services composition**. International Journal of Web and Grid Services, volume 1 (1), p.1-30. 2005.

FARRELL, Joel; LAUSEN, Holger. **Semantic Annotations for WSDL and XML Schema**. [SI], 2007. Disponível em: <http://www.w3.org/TR/sawsdl/>. Acesso em: 9 agosto 2013.

FENSEL, Dieter, et al. **Enabling Semantic Web Services: The Web Service Modeling Ontology**. Nova York: Springer, 2007.

HOBOLD, Guilherme Coan. **Uma Abordagem Automática Para Descoberta e Composição de Serviços Web Semânticos**. Dissertação de Mestrado para a obtenção do grau de Mestre em Ciência da

Computação – Universidade Federal de Santa Catarina, Florianópolis, 2012.

HOBOLD, Guilherme C.; SIQUEIRA, Frank. **Discovery of Semantic Web Services Compositions Based on SAWSDL Annotations**. International Conference on Web Services (ICWS), vol., no., pp.280-287, 2012.

JOSUTTIS, Nicolai M. **SOA na Prática**. Rio de Janeiro, RJ: Editora Alta Books, 2008.

KAPITSAKI, G., et al. **Service Composition: State of the art and future challenges**. Mobile and Wireless Communications Summit, vol., no., pp.1-5, 2007.

KREGER, Heather. **Web Service Conceptual Architecture (WSCA 1.0)**. [SI]: IBM Software Group, 2001.

KOPECKY, Jacek, et al. **SAWSDL: Semantic Annotations for WSDL and XML Schema**. IEEE Internet Computing, v. 11, n. 6, p. 60-67, 2007.

KLUSCH, Matthias; FRIES, Benedikt; SYCARA, Katia. **OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services**. Journal of Web Semantics, v. 7, n. 2, p. 121-133, 2009.

HAWKE, Sandro, et al. **Semantic Web Activity**, 2013. Disponível em: <http://www.w3.org/2001/sw/>. Acesso em: 28 fevereiro 2014.

HERMAN, Ivan. **W3C Semantic Web Frequently Asked Questions**. Disponível em: <http://www.w3.org/2001/sw/SW-FAQ>. Acesso em: 13 julho 2013.

LAGA, Nassim; BERTIN, Emmanuel; CRESPI, Noel. **User-centric Services and Service Composition, a Survey**. IEEE Software Engineering Workshop (SEW), vol., no., pp.3-9, 2008

MALAIMALAVANTHANI, M.; Gowri, R. **A Survey on Semantic Web Service Discovery**. International Conference On Information Communication And Embedded Systems, [SI], Chennai, 2013.

MALOTRA, Ashok; MALONEY, Murray. **XML Schema Requirements**. 2001. Disponível em: <http://www.w3.org/TR/1999/NOTE-xml-schema-req>. Acesso em: 06 agosto 2013.

MARTIN, David, et al. **Bringing Semantics to Web Services with OWL-S**. World Wide Web, v. 10, n. 3, p. 243-277, 2007.

MARTIN, David; DOMINGUE, John. **Semantic Web Services**. IEEE Intelligent Systems, 2007.

MCGUINESS, Deborah L.; HARMELEN, Frank van. **OWL Web Ontology Language Overview**. Disponível em: <http://www.w3.org/TR/owl-features>. Acesso em: 06 agosto 2013.

MEIYU, Pang; MAOJI, Wang. **Web services composition based on user requirement**. International Conference on Computer Science & Education (ICCSE), vol., no., pp.1481-1484, 2013.

MONTEIRO, Filipe Luiz Mélo da Costa. **Web Semântica na Automação de Composição de Web Services**. Monografia para obtenção do grau de Bacharel em Engenharia da Computação – Universidade Federal de Pernanbudo. Recife, 2008.

OWL WORKING GROUP. **Web Ontology Language (OWL)**, 2012. Disponível em: <http://www.w3.org/2001/sw/wiki/OWL>. Acesso em: 13 julho 2013.

PRAZERES, Cássio Vinicius Serafim. **Serviços Web Semânticos: da modelagem à composição.** Tese de Doutorado em Ciências – Ciências da Computação e Matemática Computacional. Instituto de Ciências Matemáticas e de Computação – ICMC. Universidade de São Paulo. 2009.

PI, Bingfeng, et al. **Flow Editor: Semantic Web Service Composition Tool**. IEEE International Conference on Services Computing (SCC), vol., no., pp.666-667, 2012.

SCHUMACHER, Michael; SCHULDT, Helko; HELIN, Helkki. **CASCOM: Inteligent Service Coordination in the Semantic Web**. Berlin: Springer, 2008.

SILVA, Diogo Phelipe Busanello; SIQUEIRA, Frank. **The Use of Semantic-based Suggestions for Web Service Composition**. International Conference on Semantic Web and Web Services (SWWS), 2014.

SYU, Yang, et al. **A Survey on Automated Service Composition Methods and Related Techniques**. IEEE International Conference on Services Computing (SCC), vol., no., pp.290-297, 2012

TIAN, Hao; LIU, Kun. **Research on semantic Web service composition**. World Automation Congress (WAC), 2012

USCHOLD, Mike; JASPER, Robert. **A Framework for Understanding and Classifying Ontology Applications**. Workshop On Ontologies And Problem-Solving Methods CEUR-WS, v. 18, p 11.1 – 11.12, Stockholm. 1999.

WEBER, Mathias. **CVFlow**. Disponível em: <http://www.lapix.ufsc.br/cvflow>. Acesso em: 06 agosto 2013.

ZHOU, Jiehan; KOIVISTO, Juha-Pekka; NIEMELÄ, Eila. **A Survey on Semantic Web Service**. Computer Supported Cooperative Work In Design, [S.I.], Nanjing, 2006.

APÊNDICE A – Ontologia books.owl

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE rdf:RDF [
<!ENTITY books.owl "http://127.0.0.1:8080/ontology/books.owl">
<!ENTITY owl "http://www.w3.org/2002/07/owl#">
<!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
<!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#">
<!ENTITY simplified_sumo.owl "http://127.0.0.1:8080/ontology/simplified_sumo.owl">
<!ENTITY xsd "http://www.w3.org/2001/XMLSchema#">
]>
<rdf:RDF xml:base="&books.owl;" xmlns="&books.owl;#"
xmlns:owl="&owl;" xmlns:rdf="&rdf;" xmlns:rdfs="&rdfs;">

<!-- Ontology Information -->
<owl:Ontology rdf:about=""
rdfs:label="Book Ontology"
owl:versionInfo="1.0">
<rdfs:comment>An ontology containing information about books</rdfs:comment>
<owl:imports>
<owl:Ontology rdf:about="&simplified_sumo.owl;"/>
</owl:imports>
</owl:Ontology>

<!-- Classes -->
<owl:Class rdf:about="#A">
<rdfs:subClassOf rdf:resource="#Grade"/>
</owl:Class>

<owl:Class rdf:about="#Article">
<rdfs:subClassOf rdf:resource="#Text"/>
</owl:Class>

<owl:Class rdf:about="#Author">
<rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>

<owl:Class rdf:about="#B">
<rdfs:subClassOf rdf:resource="#Grade"/>
</owl:Class>

<owl:Class rdf:about="#Book">
<rdfs:subClassOf rdf:resource="#Monograph"/>
<rdfs:subClassOf>
<owl:Restriction rdf:nodeID="b10">
<owl:allValuesFrom rdf:resource="#Author"/>
<owl:onProperty rdf:resource="#writtenBy"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Book-Type"/>
<owl:onProperty rdf:resource="#hasType"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Title"/>
<owl:onProperty rdf:resource="#isTitled"/>
```

```
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Book-Type"/>
<owl:Class rdf:about="#C">
<rdfs:subClassOf rdf:resource="#Grade"/>
</owl:Class>

<owl:Class rdf:about="#Comic">
<rdfs:subClassOf rdf:resource="#Genre"/>
</owl:Class>

<owl:Class rdf:about="#D">
<rdfs:subClassOf rdf:resource="#Grade"/>
</owl:Class>

<owl:Class rdf:about="#Daily">
<rdfs:subClassOf rdf:resource="#Time"/>
</owl:Class>

<owl:Class rdf:about="#Date"/>
<owl:Class rdf:about="#Encyclopedia">
<rdfs:subClassOf rdf:resource="#Book"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Very-Large"/>
<owl:onProperty rdf:resource="#hasSize"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Every-Year">
<rdfs:subClassOf rdf:resource="#Time"/>
</owl:Class>

<owl:Class rdf:about="#ExpressionContent"/>
<owl:Class rdf:about="#F">
<rdfs:subClassOf rdf:resource="#Grade"/>
</owl:Class>

<owl:Class rdf:about="#FantansyNovel">
<rdfs:subClassOf rdf:resource="#Novel"/>
</owl:Class>

<owl:Class rdf:about="#Fantasy">
<rdfs:subClassOf rdf:resource="#Genre"/>
</owl:Class>

<owl:Class rdf:about="#Genre"/>
<owl:Class rdf:about="#Grade"/>
<owl:Class rdf:about="#Hard-Cover">
<rdfs:subClassOf rdf:resource="#Book-Type"/>
</owl:Class>

<owl:Class rdf:about="#Journals">
<rdfs:subClassOf rdf:resource="#Serial-Publications"/>
</owl:Class>

<owl:Class rdf:about="#Large">
<rdfs:subClassOf rdf:resource="#Size"/>
```

```
</owl:Class>

<owl:Class rdf:about="#LinguisticExpression">
<rdfs:subClassOf rdf:resource="#ExpressionContent"/>
</owl:Class>

<owl:Class rdf:about="#Magazine">
<rdfs:subClassOf rdf:resource="#Serial-Publications"/>
</owl:Class>

<owl:Class rdf:about="#Medium">
<rdfs:subClassOf rdf:resource="#Size"/>
</owl:Class>

<owl:Class rdf:about="#Monograph">
<rdfs:subClassOf rdf:resource="#Publication"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Once"/>
<owl:onProperty rdf:resource="#timePublished"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Monthly">
<rdfs:subClassOf rdf:resource="#Time"/>
</owl:Class>

<owl:Class rdf:about="#Newspaper">
<rdfs:subClassOf rdf:resource="#Serial-Publications"/>
</owl:Class>

<owl:Class rdf:about="#Novel">
<rdfs:subClassOf rdf:resource="#Book"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Medium"/>
<owl:onProperty rdf:resource="#hasSize"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Once">
<rdfs:subClassOf rdf:resource="#Time"/>
</owl:Class>

<owl:Class rdf:about="#Paper-Back">
<rdfs:subClassOf rdf:resource="#Book-Type"/>
</owl:Class>

<owl:Class rdf:about="#Person"/>
<owl:Class rdf:about="#PrintedMaterial">
<rdfs:subClassOf rdf:resource="&simplified_sumo.owl;#Object"/>
</owl:Class>

<owl:Class rdf:about="#Publication">
<rdfs:subClassOf rdf:resource="#PrintedMaterial"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Date"/>
<owl:onProperty rdf:resource="#datePublished"/>
```

```
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Publisher"/>
<owl:onProperty rdf:resource="#publishedBy"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Publisher">
<rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>

<owl:Class rdf:about="#Reader">
<rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>

<owl:Class rdf:about="#Recommanded-Science-Fiction-Short-Story">
<rdfs:subClassOf rdf:resource="#Recommended-Short-Story"/>
<rdfs:subClassOf rdf:resource="#Science-Fiction-Short-Story"/>
</owl:Class>

<owl:Class rdf:about="#Recommended-Short-Story">
<rdfs:subClassOf rdf:resource="#ReviewedObject"/>
<rdfs:subClassOf rdf:resource="#Short-Story"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#A"/>
<owl:onProperty rdf:resource="#hasGrade"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Review"/>
<owl:Class rdf:about="#ReviewedObject">
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Grade"/>
<owl:onProperty rdf:resource="#hasGrade"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#RomanticNovel">
<rdfs:subClassOf rdf:resource="#Novel"/>
</owl:Class>

<owl:Class rdf:about="#Science-Fiction">
<rdfs:subClassOf rdf:resource="#Genre"/>
</owl:Class>

<owl:Class rdf:about="#Science-Fiction-Novel">
<rdfs:subClassOf rdf:resource="#Novel"/>
<rdfs:subClassOf rdf:resource="#ScienceFictionBook"/>
<rdfs:subClassOf>
<owl:Restriction rdf:nodeID="b6">
<owl:allValuesFrom rdf:resource="#Science-Fiction"/>
<owl:onProperty rdf:resource="#hasGenre"/>
</owl:Restriction>
</rdfs:subClassOf>
```

```
</owl:Class>

<owl:Class rdf:about="#Science-Fiction-Short-Story">
<rdfs:subClassOf rdf:resource="#Short-Story"/>
<rdfs:subClassOf rdf:nodeID="b6"/>
</owl:Class>

<owl:Class rdf:about="#ScienceFictionBook">
<rdfs:subClassOf rdf:resource="#Book"/>
</owl:Class>

<owl:Class rdf:about="#Serial-Publications">
<rdfs:subClassOf rdf:resource="#Publication"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Article"/>
<owl:onProperty rdf:resource="#contains"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Short-Story">
<rdfs:subClassOf rdf:resource="#Book"/>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Paper-Back"/>
<owl:onProperty rdf:resource="#hasType"/>
</owl:Restriction>
</rdfs:subClassOf>
<rdfs:subClassOf>
<owl:Restriction>
<owl:allValuesFrom rdf:resource="#Small"/>
<owl:onProperty rdf:resource="#hasSize"/>
</owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#Size"/>
<owl:Class rdf:about="#Small">
<rdfs:subClassOf rdf:resource="#Size"/>
</owl:Class>

<owl:Class rdf:about="#Text">
<rdfs:subClassOf rdf:resource="#LinguisticExpression"/>
<rdfs:subClassOf rdf:nodeID="b10"/>
</owl:Class>

<owl:Class rdf:about="#Thesis">
<rdfs:subClassOf rdf:resource="#PrintedMaterial"/>
</owl:Class>

<owl:Class rdf:about="#Time"/>
<owl:Class rdf:about="#Title">
<rdfs:subClassOf rdf:resource="#LinguisticExpression"/>
</owl:Class>

<owl:Class rdf:about="#User">
<rdfs:subClassOf rdf:resource="#Person"/>
</owl:Class>

<owl:Class rdf:about="#UserReview">
```

```
<rdfs:subClassOf rdf:resource="#Review"/>
</owl:Class>

<owl:Class rdf:about="#Very-Large">
<rdfs:subClassOf rdf:resource="#Size"/>
</owl:Class>

<owl:Class rdf:about="#Very-Small">
<rdfs:subClassOf rdf:resource="#Size"/>
</owl:Class>

<owl:Class rdf:about="#Weekly">
<rdfs:subClassOf rdf:resource="#Time"/>
</owl:Class>

<owl:Class rdf:about="&simplified_sumo.owl;#CorpuscularObject"/>
<owl:Class rdf:about="&simplified_sumo.owl;#Object"/>

<!-- Object Properties -->
<owl:ObjectProperty rdf:about="#contains"/>
<owl:ObjectProperty rdf:about="#datePublished"/>
<owl:ObjectProperty rdf:about="#hasGenre"/>
<owl:ObjectProperty rdf:about="#hasGrade"/>
<owl:ObjectProperty rdf:about="#hasName"/>
<owl:ObjectProperty rdf:about="#hasSize"/>
<owl:ObjectProperty rdf:about="#hasType"/>
<owl:ObjectProperty rdf:about="#isTitled"/>
<owl:ObjectProperty rdf:about="#publishedBy"/>
<owl:ObjectProperty rdf:about="#timePublished"/>
<owl:ObjectProperty rdf:about="#writtenBy"/>
<owl:ObjectProperty rdf:ID="isReserved">
<rdfs:domain rdf:resource="#Book"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>A reservation has been made by a person for a book.</rdfs:comment>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="isNotReserved">
<rdfs:domain rdf:resource="#Book"/>
<rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
>The given book is available in the library.</rdfs:comment>
</owl:ObjectProperty>
</rdf:RDF>
```